ＤＯＳ／Ｖ　ＰＣＩバス対応
補間機能付き４軸モータコントロールボード

MC8043P　取扱説明書

初版　　２００４．１１．０１
改訂　　２００７．０７．２３

NOVA electronics　　株式会社 ノヴァ電子

はじめに

このたびは、ＭＣ８０４３Ｐをご検討いただきまして、ありがとうございます。

■ 安全にお使いいただくために

本製品を安全お使いいただくために､本書に記述されている内容を必ずお守りください｡なお注意事項をお守りいただかない場合、製品の故障､瑕疵担保責任､その他一切の保証をできかねる場合があります。

本製品を使用する前に必ず本書を熟読し理解した上でご使用ください。

■ 中身をお確かめ下さい

お買い求めになった製品の添付品が揃っているかどうか確認してください｡万一､添付品が足りない場合は､すぐにお買い求めの販売店にご連絡ください。

□ボード本体	1枚
□専用Ｉ／Ｏケーブル	1本

なお、取扱説明書､ソフトウェアについては資源節約の為、添付しておりません。追加でご必要の場合はお買い求めの販売店または弊社までご請求ください。また、取扱説明書､ソフトウェアは、弊社ホームページよりダウンロードできます。
URL: http://www.novaelec.co.jp/

■ マニュアルの併用

ＭＣ８０４３Ｐの回路構成は、４軸モータコントロールＩＣ ＭＣＸ３１４Ａｓをメインとし、ＰＣＩバスのバスインターフェイス回路とＩ／Ｏインターフェイス回路から成っています。本書では、本製品及びデバイスドライバのインストール、ライブラリの使用方法、ＰＣＩバス、Ｉ／Ｏアドレス、入出力信号のインターフェイス回路を主に記述しています。モータ制御の基本機能はすべてＭＣＸ３１４Ａｓに依存していますので、これら機能動作の詳細については"ＭＣＸ３１４Ａｓ取扱説明書"を併せてご参照ください。

■ 注意･危険

引火性ガス等の近くで使用しないで下さい｡感電､火傷､焼損により大怪我や死亡につながります。

本製品は下記の環境で使用してください。

周囲温度	0～45℃
湿度(非結露)	20～90%
浮遊粉塵	特にひどくないこと
腐食性ガス	ないこと
供給電源	DC+5V(±5%)、外部電源：DC+12～24V

本製品を正しく使っていただくためにも定期的に点検を行ってください。

ケーブル接続	ボードのコネクタとケーブルが正しく接続されていること。
カードエッジ	汚れ､腐食などがないこと。
コネクタ接続部	汚れ､腐食などがないこと。
ＩＣ、ボード上	いちじるしいほこりや異物が付着していないこと。

■ 本製品の取り扱い

□本製品は静電気防止袋に入っています。本製品を取り扱う際には、人体、衣服の静電気を取り除き、基板の両端面をはさむように持つか、取付金具を持つようにしてください。

□コネクタの端子や実装部品の端子にはできるだけ触れないようにしてください。体が著しく帯電した状態でコネクタ端子や実装部品の端子に触れると、実装されているＣＭＯＳ－ＩＣを破壊する場合があります。特に冬季の乾燥した時期などは注意が必要です。

□衝撃､振動､磁気や静電気の加わる場所での保管や使用は行わないで下さい｡故障や誤動作の原因となります。

□本製品を改造しないで下さい｡改造した場合の故障、誤動作などについては一切の責任を負いません。

□供給電源が通電した状態で本製品や接続ケーブルの挿抜は行わないで下さい｡故障や誤動作の原因となります。

本書の記載内容は、今後、機能の向上などのため予告なしに変更する場合があります。最新の取扱説明書は弊社のホームページからダウンロードできます。http://www.novaelec.co.jp/

目　　次

１．概要 ---- 1
1.1 ＭＣＸ３１４Asの持つ機能の制限 ---- 2
1.2 ＭＣ８０４１Ｐとの違い ---- 2
1.3 ＰＣＩバスインターフェイス ---- 2
1.4 各軸Ｉ／Ｏインターフェイス ---- 3

２．Ｉ/Ｏアドレス設定とリード/ライトレジスタ ---- 4

３．入出力信号 ---- 5
3.1 Ｉ／Ｏコネクタ ---- 5
3.2 ドライブパルス出力信号 (nP+P,nP+N,nP-P,nP-N) ---- 6
3.3 汎用出力信号 (nOUT7,nOUT6,nOUT5,nOUT4) ---- 7
3.4 オーバランリミット入力信号 (nLMT+,nLMT-) ---- 8
3.5 減速停止／即停止入力信号 (nIN1,nIN2,nIN3) ---- 9
3.6 サーボモータ用入力信号 (nINPOS,nALARM) ---- 9
3.7 エンコーダ入力信号 (nECAP,nECAN,nECBP,nECBN,nINOP,nINON) ---- １０
3.8 外部ドライブ操作信号 (nEXOP+,nEXOP-) ---- １２
3.9 緊急停止入力信号 (EMG) ---- １２
3.10 外部電源 (VEX) ---- １２
3.11 ＰＣＩバスコネクタ ---- １３

４．割り込み ---- １５

５．モータドライバ接続例 ---- １６
5.1 ステッピングモータドライバとの接続例 ---- １６
5.2 ＡＣサーボドライバとの接続例 ---- １７

６．入出力信号タイミング ---- １８
6.1 リセット時 ---- １８
6.2 独立ドライブ開始時 ---- １８
6.3 補間ドライブ時 ---- １８
6.4 入力パルスタイミング ---- １９
■ エンコーダ２相パルス入力時 ---- １９
■ アップ/ダウンパルス入力時 ---- １９
6.5 即停止タイミング ---- １９
■ 外部信号による即停止 ---- １９
■ 命令による即停止 ---- １９
6.6 減速停止タイミング ---- ２０
■ 外部信号による減速停止 ---- ２０
■ 命令による減速速止 ---- ２０

７．基板外形 ---- ２１

８．インストール ---- ２２
8.1 ドライバソフトウェアの準備 ---- ２２
8.2 パソコンへの本ボードの組込み ---- ２２
8.3 デバイスドライバのインストール ---- ２３
8.3.1 Windows98 ---- ２３
8.3.2 Windows2000 ---- ２７
8.3.3 WindowsXP ---- ３１
8.4 取り外し ---- ３４
8.4.1 Windows98 ---- ３４
8.4.2 Windows2000/XP ---- ３４
8.5 デバイスドライバの更新 ---- ３５
8.5.1 Windows98 ---- ３５
8.5.2 Windows2000 ---- ３９
8.5.3 WindowsXP ---- ４２
8.6 外部機器との接続時の注意 ---- ４６

９．プログラミング ---- ４７
9.1 ソフトウェアの仕様 ---- ４７
9.1.1 動作環境 ---- ４７
9.1.2 プログラム構成 ---- ４７
9.1.3 ＡＰＩ（MC8043Pドライバ関数） ---- ４９
9.1.3.1 関数一覧 ---- ４９
9.1.3.2 関数仕様 ---- ５１
■補足説明 ---- ９４
9.1.3.3 使用方法 ---- １０６
9.1.4 ＡＰＩ（MC8041Pドライバで使用していた関数をサポート） ---- １１２
9.1.4.1 VC++（MC8043Pを１枚で使用する場合） ---- １１２
9.1.4.2 VC++（MC8043Pを複数枚で使用する場合 ---- １１３
9.1.4.3 VB6.0（MC8043Pを１枚で使用する場合） ---- １１４
9.1.4.4 VB6.0（MC8043Pを複数枚で使用する場合） ---- １１５
9.1.4.5 VB.NET2003（MC8043Pを１枚で使用する場合） ---- １１６
9.1.4.6 VB.NET2003（MC8043Pを複数枚で使用する場合） ---- １１７
9.1.4.7 C#（MC8043Pを１枚で使用する場合） ---- １１８
9.1.4.8 C#（MC8043Pを複数枚で使用する場合） ---- １１９
9.1.4.9 注意点 ---- １２０
9.2 ソフトウェアの内容 ---- １２１
9.3 開発手順 ---- １２６
9.3.1 VC++でアプリケーションを開発する場合（ VC++6.0, VC++.NET2003 ） ---- １２６
9.3.2 VB6.0でアプリケーションを開発する場合 ---- １２８
9.3.3 VB.NET2003でアプリケーションを開発する場合 ---- １２８
9.3.4 C#でアプリケーションを開発する場合 ---- １２９
9.4 プログラミング上の注意点 ---- １３０

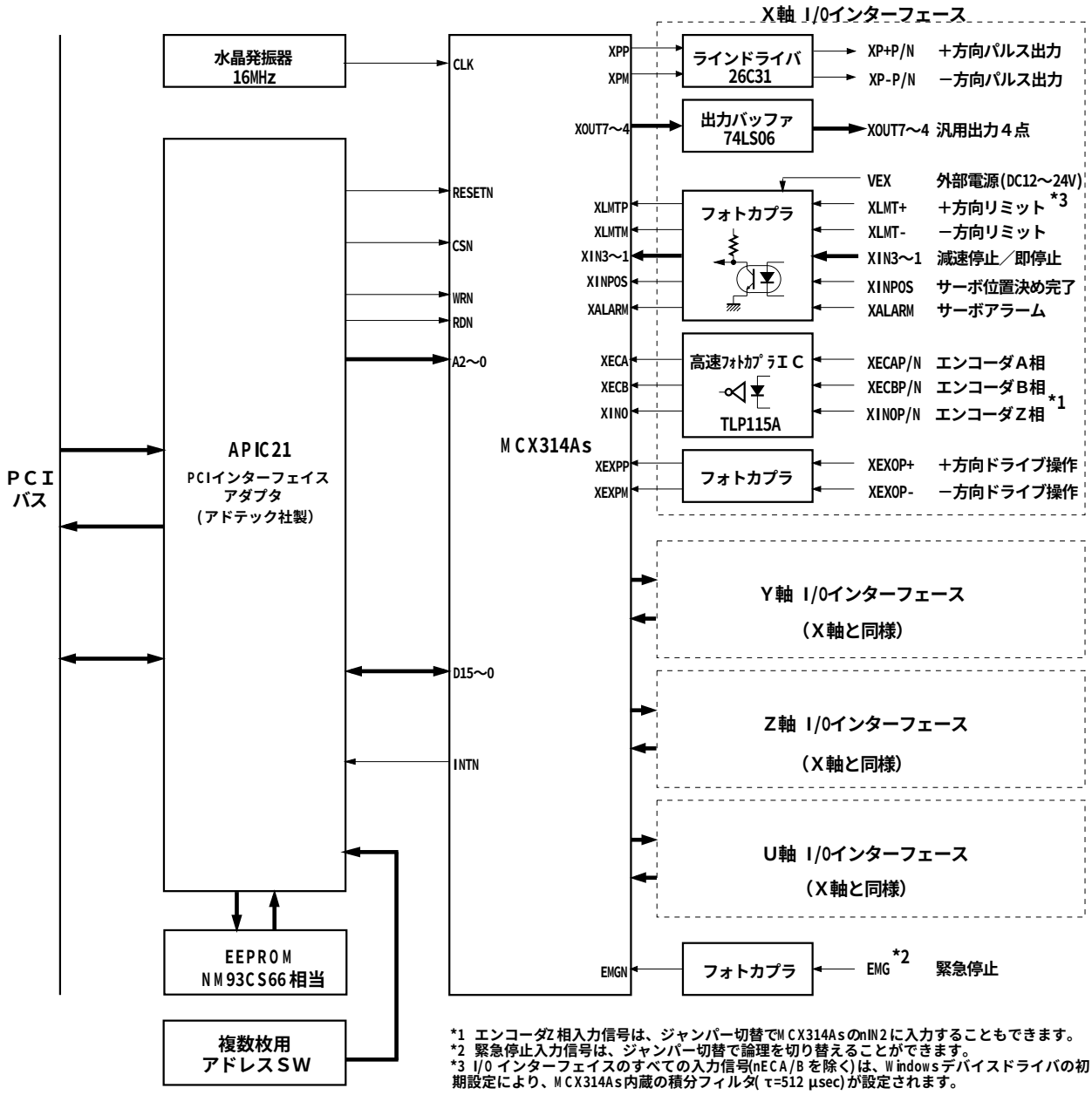
１０．評価ツール ---- １３２
10.1 概要 ---- １３２
10.2 メイン画面 ---- １３２
10.3 モード設定画面 ---- １３３
10.4 拡張モード設定画面 ---- １３３
10.5 同期動作モード設定画面 ---- １３３
10.6 ステータス画面 ---- １３４

１１．仕様まとめ ---- １３５

１．概要

ＭＣ８０４３Ｐは、補間機能付き４軸モータコントロールＩＣ ＭＣＸ３１４Ａｓを搭載した、ＰＣ/ＡＴ互換機ＰＣＩバス対応の回路基板です。１ボードで４軸のサーボモータ、またはステッピングモータを各軸独立に位置決め制御または速度制御することができます。また、４軸中、任意の２軸または３軸を選択して、２軸/３軸直線補間、ＣＷ/ＣＣＷ円弧補間、２軸/３軸ビットパターン補間（ＣＰＵからのビットデータによる補間）を行うことができます。

下図にＭＣ８０４３Ｐの機能ブロック図を示します。ＭＣ８０４３Ｐは、ＭＣＸ３１４Ａｓをメインに、ＰＣＩバスのインターフェイスと、Ｘ，Ｙ，Ｚ、Ｕ各軸のＩ／Ｏインターフェイス回路から構成されています。従って、本回路基板の基本機能はすべてＭＣＸ３１４Ａｓに依存していますので、これら機能動作の詳細についてはＭＣＸ３１４Ａｓの取扱説明書を併せてご参照ください。



ＭＣ８０４３Ｐ 回路ブロック図

1.1 ＭＣＸ３１４Ａｓの持つ機能の制限

本ボードでは、基板面積およびＩ／Ｏコネクタの端子数の制約から、MCX314Asの持つ次の入／出力信号についてはサポートしていません。

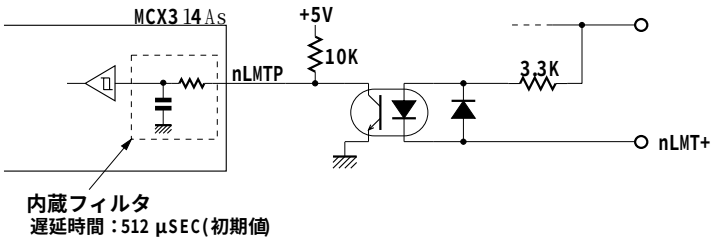
- **BUSYN出力信号**
- **EXPLSN入力信号**
- **SCLK出力信号**
- **nDIRVE/DCC出力信号**
- **nOUT3～0汎用出力信号**　（各軸のnOUT7～4の４点のみをバッファを介して出力しています。）

1.2 ＭＣ８０４１Ｐとの違い

ＭＣ８０４１ＰはＭＣＸ３１４を実装したＰＣＩボードでしたが、このＭＣ８０４３Ｐは、ＭＣＸ３１４をＭＣＸ３１４Ａｓに置き換えたＰＣＩボードです。Ｉ／Ｏインターフェイスに関しては、信号名、ピン配置などはＭＣ８０４１Ｐと全く同じですが、機能向上のため下記の項目がＭＣ８０４１Ｐと異なります。

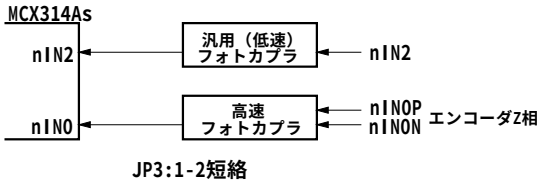
■入力信号のフィルタ回路を削除

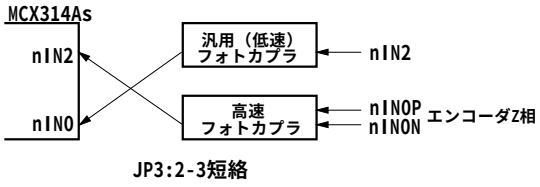
ＭＣＸ３１４Ａｓは、nECA/B信号を除く他のすべての入力信号について、ＩＣ内部に積分フィルタを内蔵しています。ＭＣ８０４３Ｐボードは、これらのＩＣ内蔵フィルタを有効利用するために、ボード上にはＣＲフィルタを実装していません。ＩＣ内蔵フィルタは、ノヴァ電子より供給されるWindowsデバイスドライバの起動時初期設定において、nECA/B信号を除く他のすべての入力信号について遅延時間512μsecに設定されます。ＩＣ内蔵フィルタは、ＭＣＸ３１４Ａｓのモード設定で自由に遅延時間を変更できます。



■エンコーダＺ相入力信号の切り替え

ＭＣ８０４１Ｐでは、エンコーダＺ相信号の入力としてnIN0を使用していました。しかし、ＭＣ８０４３Ｐに搭載されているMCX314Asの自動原点出し機能を使用する場合、nIN2の入力がエンコーダZ相の入力に割り当てられています。ＭＣ８０４３Ｐでは、ジャンパー端子JP3の1-2を短絡すると、ＭＣ８０４１Ｐ同様にnIN0がエンコーダＺ相信号として使用できます（出荷時の状態）。一方、ジャンパー端子JP3の2-3を短絡すると、エンコーダZ相入力がMCX314AsのnIN2入力に接続され、MCX314Asの自動原点出しにおいてエンコーダZ相信号を入力することができます。





1.3 ＰＣＩバスインターフェイス

■Ｉ／Ｏ占有アドレス

本ボードでは、SA15 ～ 4 をアドレスデコードし、SA3 ～ 1 で MCX314As の内部 16 ビットリード／ライトレジスタを選択します。ＰＣＩバスのＩ／０占有アドレスは、ボード当たり１６バイト使用します。本ボードのＩ／Ｏアドレス指定は、Windows 搭載のプラグアンドプレイ機能によって決定されます。

■データ長

データ長は１６ビットです。バイト単位のリード／ライトアクセスはできません。

■割り込み信号

ＰＣＩバスへの割り込みを使用する場合は、Windows 搭載のプラグアンドプレイ機能によって決定されるＩＲＱを使用します。

1.4 各軸Ｉ／Ｏインターフェイス

■ドライブパルス出力(nP+P/N,nP-P/N)

モータを駆動する＋方向／－方向のドライブパルス出力は、１PPSから最高４MPPSのデューティ５０％のパルスを出力します。各々の方向のドライブパルス出力信号は、AM26C31相当のラインドライバによる差動出力となっています。

■汎用出力(nOUT7～4)

各軸４本の汎用出力があります。出力バッファは、SN74LS06相当品を使用し、オープンコレクタ出力です。サーボモータドライバの偏差カウンタクリア、サーボフリー、アラームリセットなどに使用することができます。

■オーバランリミット入力(nLMT+,nLMT-)

＋方向、－方向のそれぞれの出力パルスを禁止する入力信号です。モード設定でアクティブ時に即停止／減速停止を選択することができます。この入力信号はフォトカプラで内部回路とは絶縁されています。外部からDC12～24Vの電源供給が必要です。

■減速停止／即停止入力(nIN3～1)

原点サーチ動作などにおいで、ドライブパルスを外部から減速停止または即停止させる入力信号です。有効／無効、アクティブ論理レベルをモード設定することができます。各軸３点用意されています。汎用入力信号としても使用することができます。この入力信号はフォトカプラで内部回路とは絶縁されています。

■サーボモータ用入力(nINPOS,nALARM)

サーボモータドライバのINPOS（位置決め完了）信号、ALARM（アラーム）信号を入力します。汎用入力信号としても使用することができます。この入力信号はフォトカプラで内部回路とは絶縁されています。

■エンコーダ入力(nECAP/N,nECBP/N,nINOP/N)

エンコーダからのＡ／Ｂ相信号、およびＺ相信号を入力します。nECAP/N,nECBP/N信号は、エンコーダのＡ／Ｂ相信号のための入力で、MCX314As内部の32ビット実位置カウンタをカウントアップ／ダウンします。nINOP/N信号はＺ相信号のための入力で、ドライブパルスを減速停止または即停止させる入力信号です。出荷時はこのnINOP/N信号はMCX314AsのnINO入力に接続されています。ジャンパー端子JP3の2-3を短絡すると、このエンコーダZ相入力がMCX314AsのnIN2入力に接続され、MCX314Asの自動原点出しを行なうことができます。この入力信号は高速フォトカプラＩＣで内部回路とは絶縁されています。差動出力のラインドライバとの接続が容易です。

■外部ドライブ操作入力(nEXOP+,nEXOP-)

外部から＋方向／－方向のドライブを起動する入力です。定量パルスドライブモードでは、入力信号のトリガ（立ち下がり）で指定ドライブパルスが出力されます。また、連続パルスドライブモードにすると、入力信号がLowレベルの間だけ、連続してドライブパルスを出し続けます。各軸のマニュアルジョグ送り等において、ＣＰＵの介在なしに応答性の速い軸送り動作が可能となります。
この入力信号はフォトカプラで内部回路とは絶縁されています。

■緊急停止入力(EMG)

全軸のドライブを緊急停止させる入力信号です。ボード上のジャンパ選択でアクティブ論理レベルを設定することができます。この入力信号はフォトカプラで内部回路とは絶縁されています。

２．Ｉ／Ｏアドレス設定とリード／ライトレジスタ

ボードのＩ／Ｏポートアドレスは、ＰＣＩバスのプラグアンドプレイ機能(以下ＰｎＰ機能)によって決定されます。1基板当たりＩ／Ｏアドレスを連続して１６バイト必要とします。
パソコンのメインボードや、他のＩ／Ｏ拡張ボードで使用しているＩ／Ｏアドレスと重ならないように[システムのプロパティ][デバイスマネージャ]で確認してください。

下表にＭＣＸ３１４Ａｓ内のリード／ライトレジスタのＩ／Ｏアドレスを示します。Ｉ／Ｏアドレスの項の（ ）内はＰｎＰ機能で０２８０～０２８Ｆhに設定されたときの、各レジスタのアドレスです。各々のレジスタは１６ビット長です。必ずワードでアクセスしてください。バイトでのアクセスはできません。各レジスタの詳細は、ＭＣＸ３１４Ａｓ取扱説明書４章を参照してください。

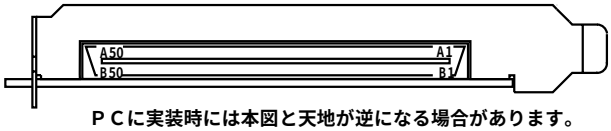
Ｉ／Ｏアドレス	ライトレジスタ		リードレジスタ	
SA3　SA2　SA1	記号	レジスタ名	記号	レジスタ名
0　0　0 （０２８０h）	WR0	コマンドレジスタ	RR0	主ステータスレジスタ
0　0　1 （０２８２h）	XWR1 YWR1 ZWR1 UWR1	Ｘ軸モードレジスタ１ Ｙ軸モードレジスタ１ Ｚ軸モードレジスタ１ Ｕ軸モードレジスタ１	XRR1 YRR1 ZRR1 URR1	Ｘ軸ステータスレジスタ１ Ｙ軸ステータスレジスタ１ Ｚ軸ステータスレジスタ１ Ｕ軸ステータスレジスタ１
0　1　0 （０２８４h）	XWR2 YWR2 ZWR2 UWR2	Ｘ軸モードレジスタ２ Ｙ軸モードレジスタ２ Ｚ軸モードレジスタ２ Ｕ軸モードレジスタ２	XRR2 YRR2 ZRR2 URR2	Ｘ軸ステータスレジスタ２ Ｙ軸ステータスレジスタ２ Ｚ軸ステータスレジスタ２ Ｕ軸ステータスレジスタ２
	BP1P	ＢＰ１Ｐレジスタ		
0　1　1 （０２８６h）	XWR3 YWR3 ZWR3 UWR3	Ｘ軸モードレジスタ３ Ｙ軸モードレジスタ３ Ｚ軸モードレジスタ３ Ｕ軸モードレジスタ３	XRR3 YRR3 ZRR3 URR3	Ｘ軸ステータスレジスタ３ Ｙ軸ステータスレジスタ３ Ｚ軸ステータスレジスタ３ Ｕ軸ステータスレジスタ３
	BP1M	ＢＰ１Ｍレジスタ		
1　0　0 （０２８８h）	WR4	アウトプットレジスタ	RR4	インプットレジスタ１
	BP2P	ＢＰ２Ｐレジスタ		
1　0　1 （０２８Ａh）	WR5	補間モードレジスタ	RR5	インプットレジスタ２
	BP2M	ＢＰ２Ｍレジスタ		
1　1　0 （０２８Ｃh）	WR6 BP3P	ライトデータレジスタ１ ＢＰ３Ｐレジスタ	RR6	リードデータレジスタ１
1　1　1 （０２８Ｅh）	WR7 BP3M	ライトデータレジスタ２ ＢＰ３Ｍレジスタ	RR7	リードデータレジスタ２

３．入出力信号

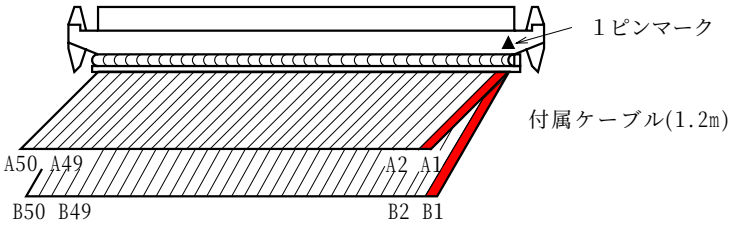
この章では、Ｉ／Ｏコネクタの各入出力信号について記述します。信号の説明、およびインターフェース回路では、各軸の信号名をｎ○○○○と記述していますが、この"ｎ"はＸ、Ｙ、Ｚ、およびＵを表しています。

3.1　Ｉ／Ｏコネクタ

Ｉ／Ｏコネクタ　ピン配置



ＰＣに実装時には本図と天地が逆になる場合があります。



付属ケーブルは、上図に示すように、コネクタの１ピンマーク（三角印）を右上にしたとき、上側ケーブルの右（赤線）から左に向かってA1,A2,… A49,A50、下側ケーブルの右（赤線）から左に向かってB1,B2,… B49,B50となります。

コネクタ型式：ボード側 FX2B-100P-1.27DS（ヒロセ），ケーブル側 FX2B-100S-1.27R（ヒロセ）

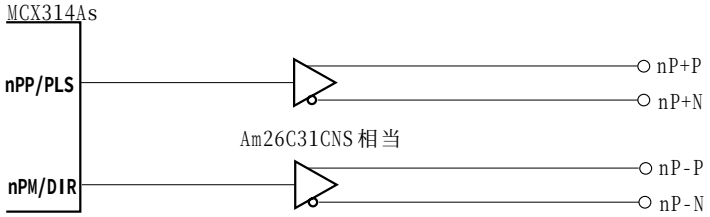
ピン	信号名	入／出	内　　容	説明	ピン	信号名	入／出	内　　容	説明
A1	VEX		外部電源（DC12～24V）	3.10	B1	VEX		外部電源（DC12～24V）	3.10
A2	EMG	入力	緊急停止（全軸共通）	3.9	B2				
A3	XLMT+	入力	Ｘ軸＋方向リミット	3.4	B3	ZLMT+	入力	Ｚ軸＋方向リミット	3.4
A4	XLMT-	入力	Ｘ軸－方向リミット	3.4	B4	ZLMT-	入力	Ｚ軸－方向リミット	3.4
A5	XIN1	入力	Ｘ軸減速停止／即停止	3.5	B5	ZIN1	入力	Ｚ軸減速停止／即停止	3.5
A6	XIN2	入力	Ｘ軸減速停止／即停止	3.5	B6	ZIN2	入力	Ｚ軸減速停止／即停止	3.5
A7	XIN3	入力	Ｘ軸減速停止／即停止	3.5	B7	ZIN3	入力	Ｚ軸減速停止／即停止	3.5
A8	YLMT+	入力	Ｙ軸＋方向リミット	3.4	B8	ULMT+	入力	Ｕ軸＋方向リミット	3.4
A9	YLMT-	入力	Ｙ軸－方向リミット	3.4	B9	ULMT-	入力	Ｕ軸－方向リミット	3.4
A10	YIN1	入力	Ｙ軸減速停止／即停止	3.5	B10	UIN1	入力	Ｕ軸減速停止／即停止	3.5
A11	YIN2	入力	Ｙ軸減速停止／即停止	3.5	B11	UIN2	入力	Ｕ軸減速停止／即停止	3.5
A12	YIN3	入力	Ｙ軸減速停止／即停止	3.5	B12	UIN3	入力	Ｕ軸減速停止／即停止	3.5
A13	XINPOS	入力	Ｘ軸サーボ位置決め完了	3.6	B13	ZINPOS	入力	Ｚ軸サーボ位置決め完了	3.6
A14	XALARM	入力	Ｘ軸サーボアラーム	3.6	B14	ZALARM	入力	Ｚ軸サーボアラーム	3.6
A15	XECAP	入力	Ｘ軸エンコーダＡ相	3.7	B15	ZECAP	入力	Ｚ軸エンコーダＡ相	3.7
A16	XECAN	入力	Ｘ軸エンコーダＡ相	3.7	B16	ZECAN	入力	Ｚ軸エンコーダＡ相	3.7
A17	XECBP	入力	Ｘ軸エンコーダＢ相	3.7	B17	ZECBP	入力	Ｚ軸エンコーダＢ相	3.7
A18	XECBN	入力	Ｘ軸エンコーダＢ相	3.7	B18	ZECBN	入力	Ｚ軸エンコーダＢ相	3.7
A19	XINOP	入力	Ｘ軸エンコーダＺ相	3.7	B19	ZINOP	入力	Ｚ軸エンコーダＺ相	3.7
A20	XINON	入力	Ｘ軸エンコーダＺ相	3.7	B20	ZINON	入力	Ｚ軸エンコーダＺ相	3.7
A21	YINPOS	入力	Ｙ軸サーボ位置決め完了	3.6	B21	UINPOS	入力	Ｕ軸サーボ位置決め完了	3.6
A22	YALARM	入力	Ｙ軸サーボアラーム	3.6	B22	UALARM	入力	Ｕ軸サーボアラーム	3.6
A23	YECAP	入力	Ｙ軸エンコーダＡ相	3.7	B23	UECAP	入力	Ｕ軸エンコーダＡ相	3.7
A24	YECAN	入力	Ｙ軸エンコーダＡ相	3.7	B24	UECAN	入力	Ｕ軸エンコーダＡ相	3.7
A25	YECBP	入力	Ｙ軸エンコーダＢ相	3.7	B25	UECBP	入力	Ｕ軸エンコーダＢ相	3.7
A26	YECBN	入力	Ｙ軸エンコーダＢ相	3.7	B26	UECBN	入力	Ｕ軸エンコーダＢ相	3.7
A27	YINOP	入力	Ｙ軸エンコーダＺ相	3.7	B27	UINOP	入力	Ｕ軸エンコーダＺ相	3.7
A28	YINON	入力	Ｙ軸エンコーダＺ相	3.7	B28	UINON	入力	Ｕ軸エンコーダＺ相	3.7
A29	XEXOP+	入力	Ｘ軸＋方向ドライブ操作	3.8	B29	ZEXOP+	入力	Ｚ軸＋方向ドライブ操作	3.8
A30	XEXOP-	入力	Ｘ軸－方向ドライブ操作	3.8	B30	ZEXOP-	入力	Ｚ軸－方向ドライブ操作	3.8
A31	YEXOP+	入力	Ｙ軸＋方向ドライブ操作	3.8	B31	UEXOP+	入力	Ｕ軸＋方向ドライブ操作	3.8
A32	YEXOP-	入力	Ｙ軸－方向ドライブ操作	3.8	B32	UEXOP-	入力	Ｕ軸－方向ドライブ操作	3.8
A33	GND		内部回路ＧＮＤ		B33	GND		内部回路ＧＮＤ	
A34	XOUT4	出力	Ｘ軸汎用出力	3.3	B34	ZOUT4	出力	Ｚ軸汎用出力	3.3
A35	XOUT5	出力	Ｘ軸汎用出力	3.3	B35	ZOUT5	出力	Ｚ軸汎用出力	3.3

ピン	信号名	入／出	内　　容	説明	ピン	信号名	入／出	内　　容	説明
A36	XOUT6	出力	Ｘ軸汎用出力	3.3	B36	ZOUT6	出力	Ｚ軸汎用出力	3.3
A37	XOUT7	出力	Ｘ軸汎用出力	3.3	B37	ZOUT7	出力	Ｚ軸汎用出力	3.3
A38	XP+P	出力	Ｘ軸＋方向ドライブパルス	3.2	B38	ZP+P	出力	Ｚ軸＋方向ドライブパルス	3.2
A39	XP+N	出力	Ｘ軸＋方向ドライブパルス	3.2	B39	ZP+N	出力	Ｚ軸＋方向ドライブパルス	3.2
A40	XP-P	出力	Ｘ軸－方向ドライブパルス	3.2	B40	ZP-P	出力	Ｚ軸－方向ドライブパルス	3.2
A41	XP-N	出力	Ｘ軸－方向ドライブパルス	3.2	B41	ZP-N	出力	Ｚ軸－方向ドライブパルス	3.2
A42	GND		内部回路ＧＮＤ		B42	GND		内部回路ＧＮＤ	
A43	YOUT4	出力	Ｙ軸汎用出力	3.3	B43	UOUT4	出力	Ｕ軸汎用出力	3.3
A44	YOUT5	出力	Ｙ軸汎用出力	3.3	B44	UOUT5	出力	Ｕ軸汎用出力	3.3
A45	YOUT6	出力	Ｙ軸汎用出力	3.3	B45	UOUT6	出力	Ｕ軸汎用出力	3.3
A46	YOUT7	出力	Ｙ軸汎用出力	3.3	B46	UOUT7	出力	Ｕ軸汎用出力	3.3
A47	YP+P	出力	Ｙ軸＋方向ドライブパルス	3.2	B47	UP+P	出力	Ｕ軸＋方向ドライブパルス	3.2
A48	YP+N	出力	Ｙ軸＋方向ドライブパルス	3.2	B48	UP+N	出力	Ｕ軸＋方向ドライブパルス	3.2
A49	YP-P	出力	Ｙ軸－方向ドライブパルス	3.2	B49	UP-P	出力	Ｕ軸－方向ドライブパルス	3.2
A50	YP-N	出力	Ｙ軸－方向ドライブパルス	3.2	B50	UP-N	出力	Ｕ軸－方向ドライブパルス	3.2

注意：Ｉ/Ｏコネクタにケーブルを挿抜する場合は、まずパソコンの電源をＯＦＦの状態にし、ケーブルに供給している外部電源（ＤＣ＋２４Ｖ等）をＯＦＦにしてから挿抜して下さい。尚、ケーブルを挿入する場合はコネクタの向きに注意し、逆挿しにならないようにして下さい。パソコンや外部電源がＯＮのまま接続した場合、基板の内部回路等が破損する場合があります。

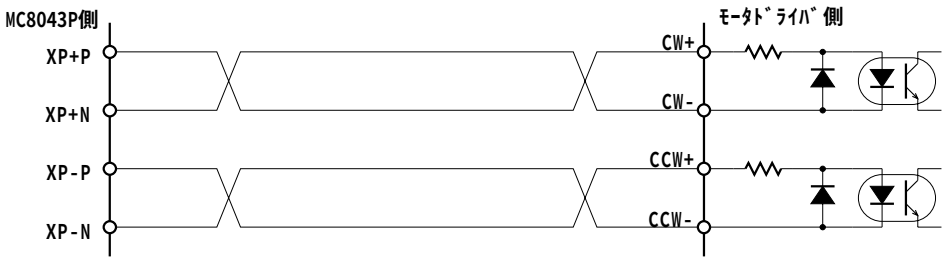
3.2　ドライブパルス出力信号（nP+P,nP+N,nP-P,nP-N）

ドライブパルス出力信号は、MCX314Asの＋方向／－方向のドライブパルス信号を差動出力のラインドライバ（AM26C31相当）を介して出力しています。nP+NはnP+Pの反転出力、nP-NはnP-Pの反転出力です。リセット時には、正出力（nP+P,nP-P）がLowレベル、反転出力（nP+N,nP-N）がHiレベルになっています。
ドライブパルス出力は、リセット直後は＋／－方向の独立２パルス方式になっていますが、モード設定によって方向・１パルス方式にすることもできます。MCX314As取扱説明書2.9.2節、4.5節を参照してください。

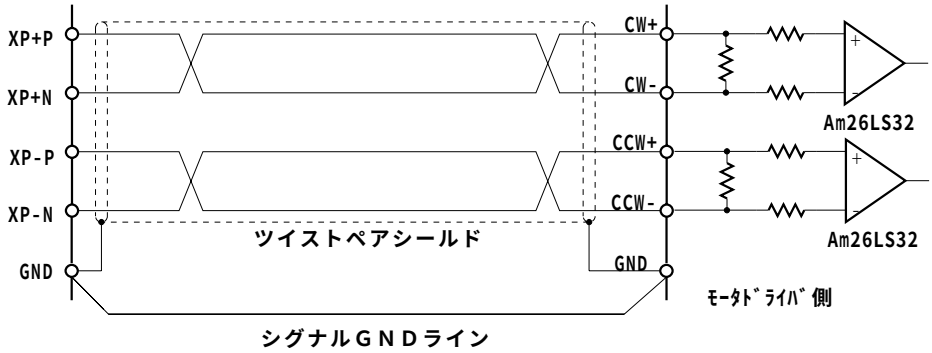


ドライブパルス出力信号回路

下図にフォトカプラ入力回路およびラインレシーバ入力回路を持つモータドライバとの接続例を示します。



フォトカプラ入力回路のモータドライバとの接続例

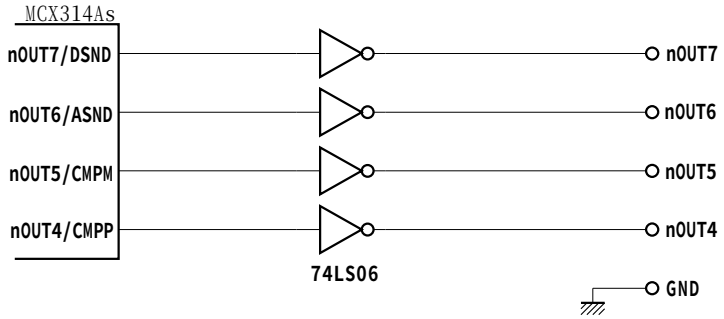


ラインレシーバ入力回路のモータドライバとの接続例

注意：ラインレシーバ入力回路を使用する場合の注意
上記のように、ラインレシーバ入力回路を使用する場合は、ラインドライバ側とモータドライバ側をシグナルＧＮＤラインで接続して下さい。機器間のシグナルＧＮＤに電位差がありますと、誤動作やドライバ回路、モータドライバ回路が損傷することがあります。上図のようにシグナルＧＮＤを別途結んでご使用下さい。

3.3 汎用出力信号（nOUT7,nOUT6,nOUT5,nOUT4）

汎用出力信号は、MCX314AsのnOUT7/DSND,nOUT6/ASND,nOUT5/CMPM,nOUT4/CMPP信号を、バッファ（74LS06）を介して出力しています。
リセット時には、各出力ともＯＦＦ状態になっています。

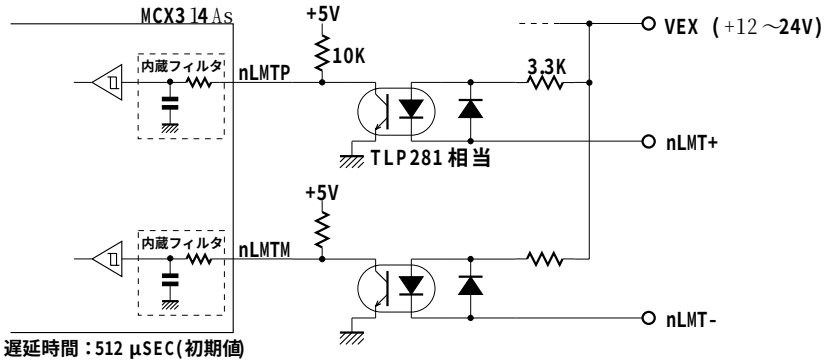


汎用出力信号回路

汎用出力信号は、モータドライバの偏差カウンタクリア、アラームリセット、励磁ＯＦＦ信号などに使用することができます。
また、モード選択でドライブ加速中／減速中、位置カウンタとコンペアレジスタの大小状態を出力することができます。
汎用出力信号の設定については、MCX314As取扱説明書2.9.8節、4.6節を参照してください。また、ドライブ加速中／減速中出力については2.9.7節、4.6節を、位置カウンタとコンペアレジスタの大小状態出力については2.3節、4.6節を参照してください。

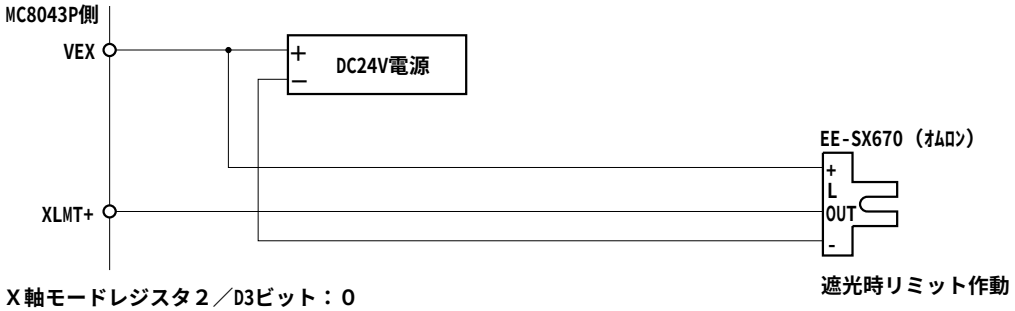
3.4 オーバランリミット入力信号（nLMT+,nLMT-）

＋方向、－方向のそれぞれのドライブパルスを抑止する入力信号です。この入力信号はフォトカプラを通してMCX314Asのリミット入力に接続されています。リセット直後は、MCX314AsはLowレベルでアクティブになりますので、信号端子（nLMT+,nLMT-）より電流が流出するときリミット機能が作動します。論理レベル、即停止／減速停止を変更することができます。モード設定の詳細は、MCX314As取扱説明書4.5節を参照してください。
この信号を動作させるには、外部からDC12～24Vの電源供給が必要です。MCX314Asに内蔵されている積分フィルタは、ノヴァ電子より供給されるWindowsデバイスドライバの起動時初期設定において、遅延時間512μsecに設定されます。システムのノイズ環境によって、この信号遅延時間を変更することができます。詳細は、MCX314As取扱説明書2.8節、6.16節を参照してください。



オーバランリミット入力信号回路

下図にオーバランリミット入力信号をフォトマイクロセンサに接続する例を示します。Ｘ軸のモードレジスタ２（XWR2）のD3ビットを０（リセット時のモード）にすると、遮光時にリミット機能が作動します。



オーバランリミット入力信号とフォトマイクロセンサとの接続例

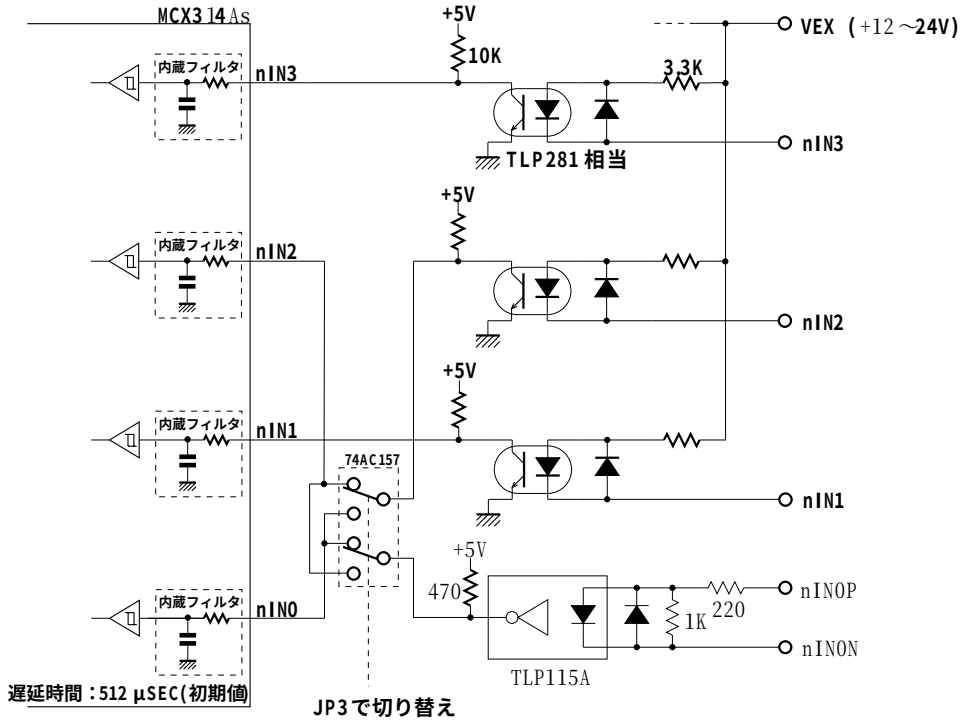
配線を長く引き回す場合は、シールド線を使用してください。

3.5 減速停止／即停止入力信号（nIN1,nIN2,nIN3）

ドライブパルス出力を途中で減速停止または即停止させるための３点の入力信号です。MCX314Asは各軸ともIN3～IN0の４点持っています。ジャンパJP3の1-2を短絡（出荷時の状態）すると、MCX314AsのnIN0にはエンコーダZ相のためのインターフェイス回路（高速フォトカプラTLP115A）が組まれます。nIN1,nIN2,nIN3信号は原点、原点近傍などの入力信号として使用します。JP3の2-3を短絡すると、エンコーダZ相のためのインターフェイス回路（高速フォトカプラTLP115A）がMCX314AsのnIN2に接続されます。この接続にするとMCX314Asのもつ自動原点出し機能を使用することができます。自動原点出しの詳細はMCX314As取扱説明書2.5節を参照してください。

それぞれの入力信号は、有効／無効、論理レベルをモード設定することができます。有効にモード設定すると、ドライブの途中で本信号がアクティブになるとドライブパルス出力を停止します。加減速ドライブ中であれば減速停止、定速ドライブ中であれば即停止します。リセット直後は、全信号が無効になっています。例えば、Ｘ軸IN3信号において、XWR1レジスタD7,D6ビットを1,0にセットし、Lowレベルで有効にすると、本ボードのXIN3信号端子より電流が流出するとドライブが停止します。モード設定の詳細は、MCX314As取扱説明書4.4節を参照してください。

この信号を動作させるには、外部からDC12～24Vの電源供給が必要です。また、この信号はインプットレジスタ１，２（RR4,5)で信号状態を常時読み出せますので汎用入力としても使用することができます。また、下図に示すMCX314As内蔵の積分フィルタは、ノヴァ電子より供給されるWindowsデバイスドライバの起動時初期設定において、遅延時間512μsecに設定されます。システムのノイズ環境によって、この信号遅延時間を変更することができます。詳細は、MCX314As取扱説明書2.8節、6.16節を参照してください。



JP3：nIN0/nIN2切替え

JP3 1-2短絡	ノーマル (出荷時)	基板I/O コネクタnINOP/N 信号がMCX314AsのnIN0に接続され、 基板I/O コネクタnIN2信号がMCX314AsのnIN2に接続されます。
JP3 2-3短絡	クロス	基板I/O コネクタnINOP/N 信号がMCX314AsのnIN2に接続され、 基板I/O コネクタnIN2信号がMCX314AsのnIN0に接続されます。

減速停止／即停止入力信号回路

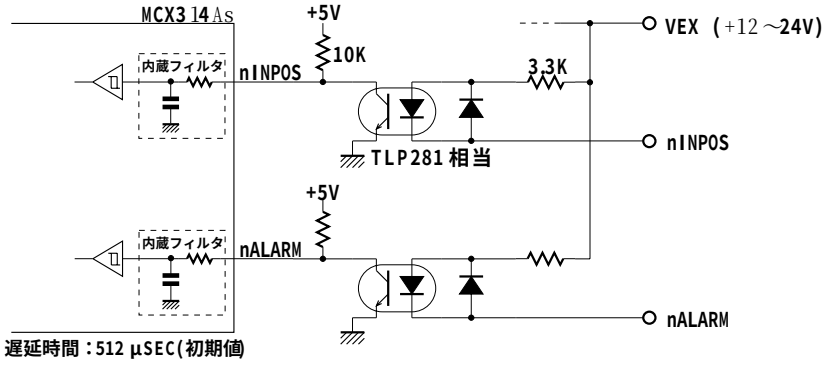
3.6 サーボモータ用入力信号（nINPOS,nALARM）

nINPOS入力信号はサーボモータドライバのインポジション（位置決め完了）出力に対応する入力です。MCX314Asのモード設定で有効／無効、論理レベルを選択します。有効に設定すると、ドライブ終了後、この信号がアクティブになるのを待ってから主ステータスレジスタ(RR0)のn-DRVビットが０に戻ります。

nALARM入力信号はサーボモータドライバのアラーム出力に対応します。モード設定で有効／無効、論理レベルを選択します。有効に設定すると、nALARM入力信号を常に監視し、アクティブ状態の場合はステータスレジスタ２(nRR2)のALARMビットに１が立ちます。ドライブ中にアクティブレベルになると、ドライブは即停止されます。

リセット直後は、両信号とも無効になっています。nINPOS入力信号については、MCX314Asのモードレジスタ２(nWR2)のD15,14ビットを1,0にセットし、Lowレベルで有効にすると、本ボードのnINPOS信号端子より電流が流出する状態を待ってから、RR0レジス

タのn-DRVビットが０に戻ります。また、nALARM入力信号については、nWR2レジスタのD13,12ビットを1,0にセットし、Lowレベルで有効にすると、本ボードのnALARM信号端子より電流が流出するときアラーム状態になります。詳細は、MCX314As取扱説明書の2.9.5節、4.5節を参照してください。



サーボモータ用入力信号回路

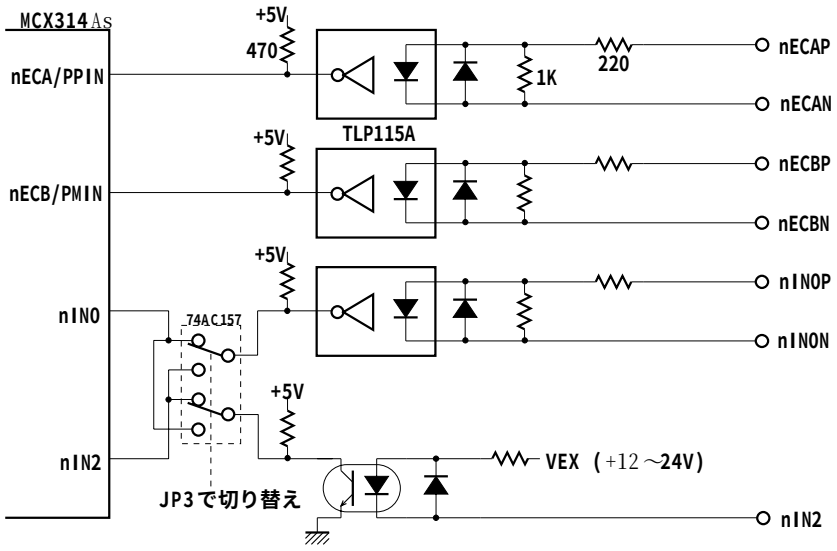
この信号を動作させるには、外部からDC12～24Vの電源供給が必要です。また、上図に示すMCX314As内蔵の積分フィルタは、ノヴァ電子より供給されるWindowsデバイスドライバの起動時初期設定において、遅延時間512μsecに設定されます。システムのノイズ環境によって、この信号遅延時間を変更することができます。詳細は、MCX314As取扱説明書2.8節、6.16節を参照してください。

サーボモータ用入力信号はインプットレジスタ１，２（RR4,5)で信号状態を常時読み出せますので汎用入力としても使用することができます。

3.7 エンコーダ入力信号（nECAP,nECAN,nECBP,nECBN,nINOP,nINON）

nECAP/N,nECBP/N入力信号はエンコーダの２相出力信号、またはサーボモータドライバのエンコーダ２相出力信号を接続して、MCX314Asの実位置カウンタをカウントするための入力です。詳細は、MCX314As取扱説明書の2.3.1節、2.9.3節、4.5節を参照してください。

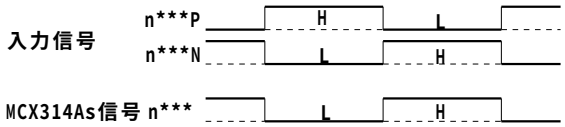
nINOP/N入力信号は、エンコーダ、またはサーボモータドライバのＺ相出力信号を接続して、ドライブパルス出力を途中で停止させるための入力です。有効／無効、論理レベルをモード設定することができます。有効にモード設定すると、ドライブの途中で本信号がアクティブになるとドライブパルス出力を停止します。3.5節でも記述していますが、JP3の2-3を短絡すると、エンコーダＺ相のためのインターフェイス回路（高速フォトカプラTLP115A）がMCX314AsのnIN2に接続されます。この接続にするとMCX314Asのもつ自動原点出し機能を使用することができます。自動原点出しの詳細はMCX314As取扱説明書2.5節を参照してください。



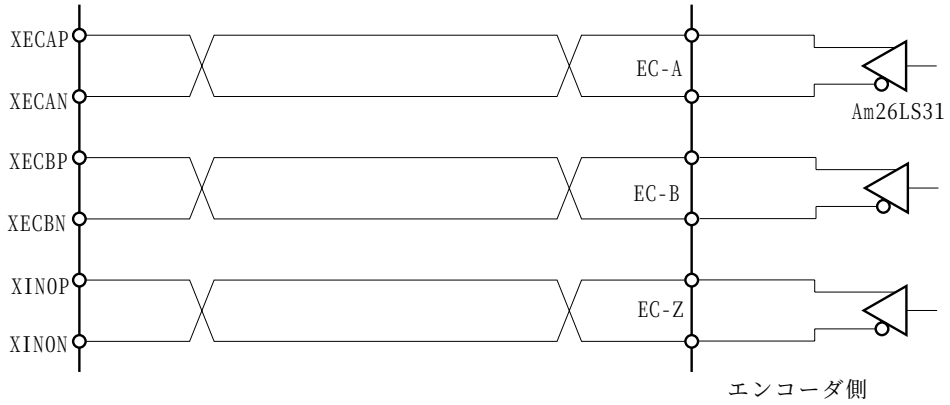
エンコーダ入力信号回路

エンコーダ入力信号回路は、上図に示すように、高速フォトカプラＩＣ TLP115A（東芝）を使用しています。各入力信号は差動出力のラインドライバとの直結が可能です。下図に示すように、n***P/N信号がH/LのときMCX314Asのn***信号がLowになり、L/HのときHiになります。入力からMCX314As信号端子までの遅延時間は100nSEC以下ですので、２相パルス入力の場合であれば最高４

MHzまでカウントできます。

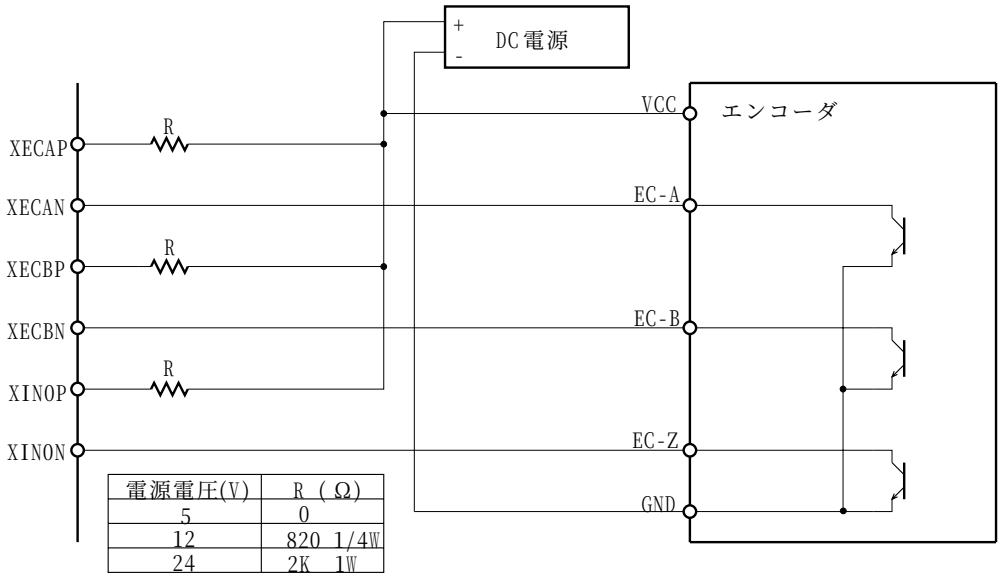


下図にエンコーダ入力信号と差動出力のラインドライバとの接続例を示します。



差動出力のラインドライバとの接続例

下図はエンコーダ入力信号とオープンコレクタ出力のエンコーダとの接続例です。



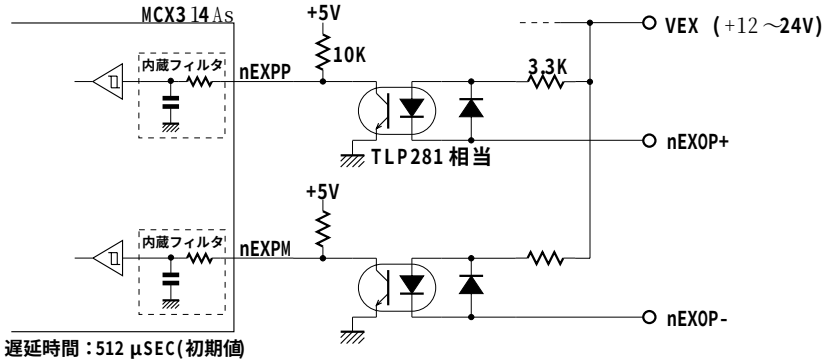
電源電圧(V)	R (Ω)
5	0
12	820 1/4W
24	2K 1W

オープンコレクタ出力との接続例

3.8 外部ドライブ操作信号（nEXOP+,nEXOP-）

外部から＋方向／－方向のドライブを起動する入力です。定量パルスドライブモードでは、入力信号のトリガ（立ち下がり）で指定ドライブパルスが出力されます。また、連続パルスドライブモードにすると、入力信号がLowレベルの間だけ、連続してドライブパルスを出し続けます。各軸のマニュアルジョグ送り等において、ＣＰＵの介在なしに軸送り動作が可能となります。外部ドライブ信号を有効にするには、MCX314Asのモード設定が必要です。詳細は、MCX314As取扱説明書の2.9.1節、4.6節を参照してください。

この信号を動作させるには、外部からDC12～24Vの電源供給が必要です。また、下図に示すMCX314As内蔵の積分フィルタは、ノヴァ電子より供給されるWindowsデバイスドライバの起動時初期設定において、遅延時間512μsecに設定されます。システムのノイズ環境によって、この信号遅延時間を変更することができます。詳細は、MCX314As取扱説明書2.8節、6.16節を参照してください。

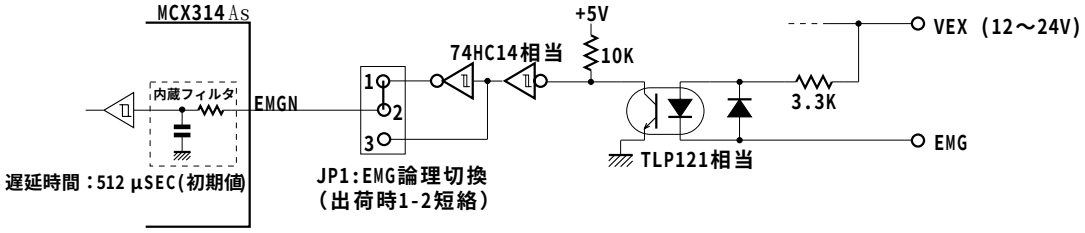


外部ドライブ操作信号回路

3.9 緊急停止入力信号（EMG）

緊急停止信号がアクティブレベルになると全軸のドライブパルス出力が停止します。アクティブレベルはボード内のＪＰ１ジャンパー端子で切り替えることができます。ドライブ中に緊急停止信号がアクティブになると、すべての軸のドライブは即停止し、主ステータスレジスタの全軸のエラービットに１が立ちます。MCX314Asの緊急停止については、MCX314As取扱説明書の2.9.6節、4.12節を参照してください。

JP1:1-2短絡　　緊急停止信号(EMG)が外部電源のＧＮＤと短絡状態になるとアクティブレベルになります。
JP1:2-3短絡　　緊急停止信号(EMG)がオープン状態になるとアクティブレベルになります。
出荷時は、1-2短絡になっています。



緊急停止入力信号回路

緊急停止信号を動作させるには、外部からDC12～24Vの電源供給が必要です。また、上図に示すMCX314As内蔵の積分フィルタは、ノヴァ電子より供給されるWindowsデバイスドライバの起動時初期設定において、遅延時間512μsecに設定されます。システムのノイズ環境によって、この信号遅延時間を変更することができます。詳細は、MCX314As取扱説明書2.8節、6.16節を参照してください。

3.10 外部電源（VEX）

外部電源は、各軸のオーバランリミット入力信号（nLMT+,nLMT-）、減速停止／即停止入力信号（nIN1,nIN2,nIN3）、サーボモータ用入力信号（nINPOS,nALARM）、外部ドライブ操作信号（nEXOP+,nEXOP-）、および緊急停止入力信号（EMG）を動作させるために、外部から供給する電源です。ＤＣ１２Ｖ～２４Ｖの範囲の電源を供給してください。入力信号１点あたりの消費電流は、ＤＣ１２Ｖの場合３．３ｍＡ、ＤＣ２４Ｖの場合７ｍＡです。

3.11　ＰＣＩバスコネクタ

ピン	信号名	内　　容	信号出力時の駆動方法	入出力方向	
				マスタ	ターゲット
A1	TRST#	テストリセット		入力	入力
A2	+12V	電源		入力	入力
A3	TMS	テストモードセレクト		入力	入力
A4	TDI	テストデータ入力		入力	入力
A5	+5V	電源		入力	入力
A6	INTA#	割り込み要求Ａ	オープン･ドレイン	出力	出力
A7	INTC#	割り込み要求Ｃ	オープン･ドレイン	出力	出力
A8	+5V	電源		入力	入力
A9		予約			
A10	+5V	電源		入力	入力
A11		予約			
A12	GND	グランド		入力	入力
A13	GND	グランド		入力	入力
A14		予約			
A15	RST#	リセット		入力	入力
A16	+5V	電源		入力	入力
A17	GNT#	グラント	トライ･ステート	入力	
A18	GND	グランド		入力	入力
A19		予約			
A20	AD30	アドレスデータ30	トライ･ステート	入/出	入/出
A21	+3.3V	電源		入力	入力
A22	AD28	アドレスデータ28	トライ･ステート	入/出	入/出
A23	AD26	アドレスデータ26	トライ･ステート	入/出	入/出
A24	GND	グランド		入力	入力
A25	AD24	アドレスデータ24	トライ･ステート	入/出	入/出
A26	IDSEL	イニシャライゼーション デバイスセレクト		入力	入力
A27	+3.3V	電源		入力	入力
A28	AD22	アドレスデータ22	トライ･ステート	入/出	入/出
A29	AD20	アドレスデータ20	トライ･ステート	入/出	入/出
A30	GND	グランド		入力	入力
A31	AD18	アドレスデータ18	トライ･ステート	入/出	入/出
A32	AD16	アドレスデータ16	トライ･ステート	入/出	入/出
A33	+3.3V	電源		入力	入力
A34	FRAME#	サイクルフレーム	サスティンド･トライ･ステート	出力	入力
A35	GND	グランド		入力	入力
A36	TRDY#	ターゲットレディ	サスティンド･トライ･ステート	入力	出力
A37	GND	グランド		入力	入力
A38	STOP#	ストップ	サスティンド･トライ･ステート	入力	出力
A39	+3.3V	電源		入力	入力
A40	SDONE	スヌープ完了		入/出	入/出
A41	SB0#	スヌープバックオフ		入/出	入/出
A42	GND	グランド		入力	入力
A43	PAR	パリティ	トライ･ステート	入/出	入/出
A44	AD15	アドレスデータ15	トライ･ステート	入/出	入/出
A45	+3.3V	電源		入力	入力
A46	AD13	アドレスデータ13	トライ･ステート	入/出	入/出
A47	AD11	アドレスデータ11	トライ･ステート	入/出	入/出
A48	GND	グランド		入力	入力
A49	AD9	アドレスデータ9	トライ･ステート	入/出	入/出
A50		キーウェイ			
A51		キーウェイ			
A52	C/BE0#	バスコマンド･バイトイネーブル0	トライ･ステート	出力	入力
A53	+3.3V	電源		入力	入力
A54	AD6	アドレスデータ6	トライ･ステート	入/出	入/出
A55	AD4	アドレスデータ4	トライ･ステート	入/出	入/出
A56	GND	グランド		入力	入力
A57	AD2	アドレスデータ2	トライ･ステート	入/出	入/出
A58	AD0	アドレスデータ0	トライ･ステート	入/出	入/出
A59	+5V	電源		入力	入力
A60	REQ64#	64ビット転送要求	サスティンド･トライ･ステート	出力	入力
A61	+5V	電源		入力	入力
A62	+5V	電源		入力	入力

ピン	信号名	内容	信号出力時の駆動方法	入出力方向	
				マスタ	ターゲット
B1	-12V	電源		入力	入力
B2	TCK	テストクロック		入力	入力
B3	GND	グランド		入力	入力
B4	TDO	テストデータ出力		出力	出力
B5	+5V	電源		入力	入力
B6	+5V	電源		入力	入力
B7	INTB#	割り込み要求B	オープン･ドレイン	出力	出力
B8	INTD#	割り込み要求D	オープン･ドレイン	出力	出力
B9	PRSNT1#				
B10		予約			
B11	PRSNT2#				
B12	GND	グランド		入力	入力
B13	GND	グランド		入力	入力
B14		予約			
B15	GND	グランド		入力	入力
B16	CLK	クロック		入力	入力
B17	GND	グランド		入力	入力
B18	REQ#	リクエスト	トライ･ステート	出力	
B19	+5V	電源		入力	入力
B20	AD31	アドレスデータ31	トライ･ステート	入/出	入/出
B21	AD29	アドレスデータ29	トライ･ステート	入/出	入/出
B22	GND	グランド		入力	入力
B23	AD27	アドレスデータ27	トライ･ステート	入/出	入/出
B24	AD25	アドレスデータ25	トライ･ステート	入/出	入/出
B25	+3.3V	電源		入力	入力
B26	C/BE3#	バスコマンド･バイトイネーブル3	トライ･ステート	出力	入力
B27	AD23	アドレスデータ23	トライ･ステート	入/出	入/出
B28	GND	グランド		入力	入力
B29	AD21	アドレスデータ21	トライ･ステート	入/出	入/出
B30	AD19	アドレスデータ19	トライ･ステート	入/出	入/出
B31	+3.3V	電源		入力	入力
B32	AD17	アドレスデータ17	トライ･ステート	入/出	入/出
B33	C/BE2#	バスコマンド･バイトイネーブル2	トライ･ステート	出力	入力
B34	GND	グランド		入力	入力
B35	IRDY#	イニシエータ･レディ	サスティンド･トライ･ステート	出力	入力
B36	+3.3V	電源		入力	入力
B37	DEVSEL#	デバイス･セレクト	サスティンド･トライ･ステート	入力	出力
B38	GND	グランド		入力	入力
B39	LOCK#	ロック	サスティンド･トライ･ステート	出力	入力
B40	PERR#	パリティ･エラー	サスティンド･トライ･ステート	入/出	入力
B41	+3.3V	電源		入力	入力
B42	SERR#	システム･エラー	オープン･ドレイン	出力	出力
B43	+3.3V	電源		入力	入力
B44	C/BE1#	バスコマンド･バイトイネーブル1	トライ･ステート	出力	入力
B45	AD14	アドレスデータ14	トライ･ステート	入/出	入/出
B46	GND	グランド		入力	入力
B47	AD12	アドレスデータ12	トライ･ステート	入/出	入/出
B48	AD10	アドレスデータ10	トライ･ステート	入/出	入/出
B49	GND	グランド		入力	入力
B50		キーウェイ			
B51		キーウェイ			
B52	AD8	アドレスデータ8	トライ･ステート	入/出	入/出
B53	AD7	アドレスデータ7	トライ･ステート	入/出	入/出
B54	+3.3V	電源		入力	入力
B55	AD5	アドレスデータ5	トライ･ステート	入/出	入/出
B56	AD3	アドレスデータ3	トライ･ステート	入/出	入/出
B57	GND	グランド		入力	入力
B58	AD1	アドレスデータ1	トライ･ステート	入/出	入/出
B59	+5V	電源		入力	入力
B60	ACK64#	64ビット転送アクノリッジ	サスティンド･トライ･ステート	入力	出力
B61	+5V	電源		入力	入力
B62	+5V	電源		入力	入力

信号名に#の付いている信号は負論理を表します。

４．割り込み

本ボードでは、MCX314Asで発生する割り込み信号を、ＰＣＩバスの4本の割り込み要求信号の内、ＩＮＴＡ＃に接続しています。Windows上のアプリケーションプログラムにおいて、割り込み処理を行なうことが可能です。

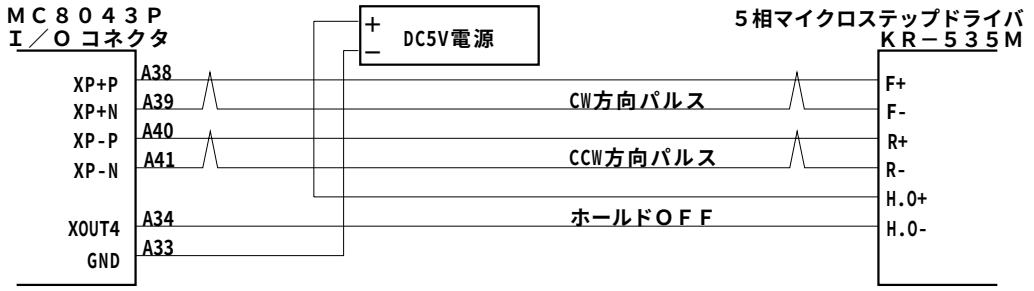
アプリケーションプログラムはVCで作成してください。VBプログラムでの割り込みはできません。

割り込みを使用するプログラミングの詳細は、9章プログラミングを参照してください。

５．モータドライバ接続例

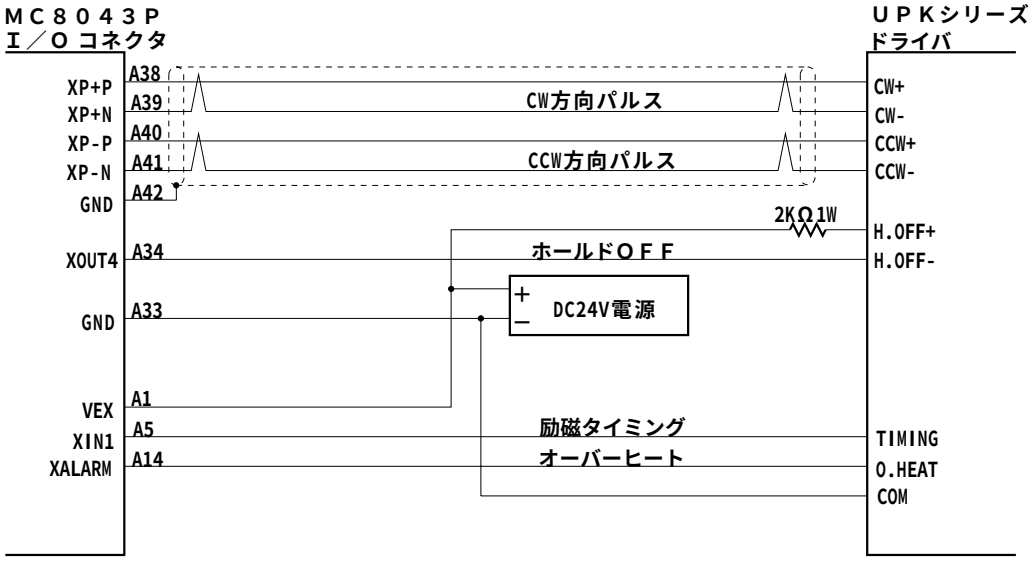
5.1 ステッピングモータドライバとの接続例

下図は、MC8043PのＸ軸とテクノドライブ製の５相マイクロステップドライバＫＲ５３５Ｍとの接続例を示しています。



注１：ホールドＯＦＦは必要に応じて配線します。ホールドＯＦＦ信号は、MCX314AsのWR3レジスタのD8ビットに0,1を書き込むことによって制御します。

下図は、MC8043PのＸ軸とオリエンタルモータ製ＵＰＫシリーズのステッピングモータドライバとの接続例を示しています。

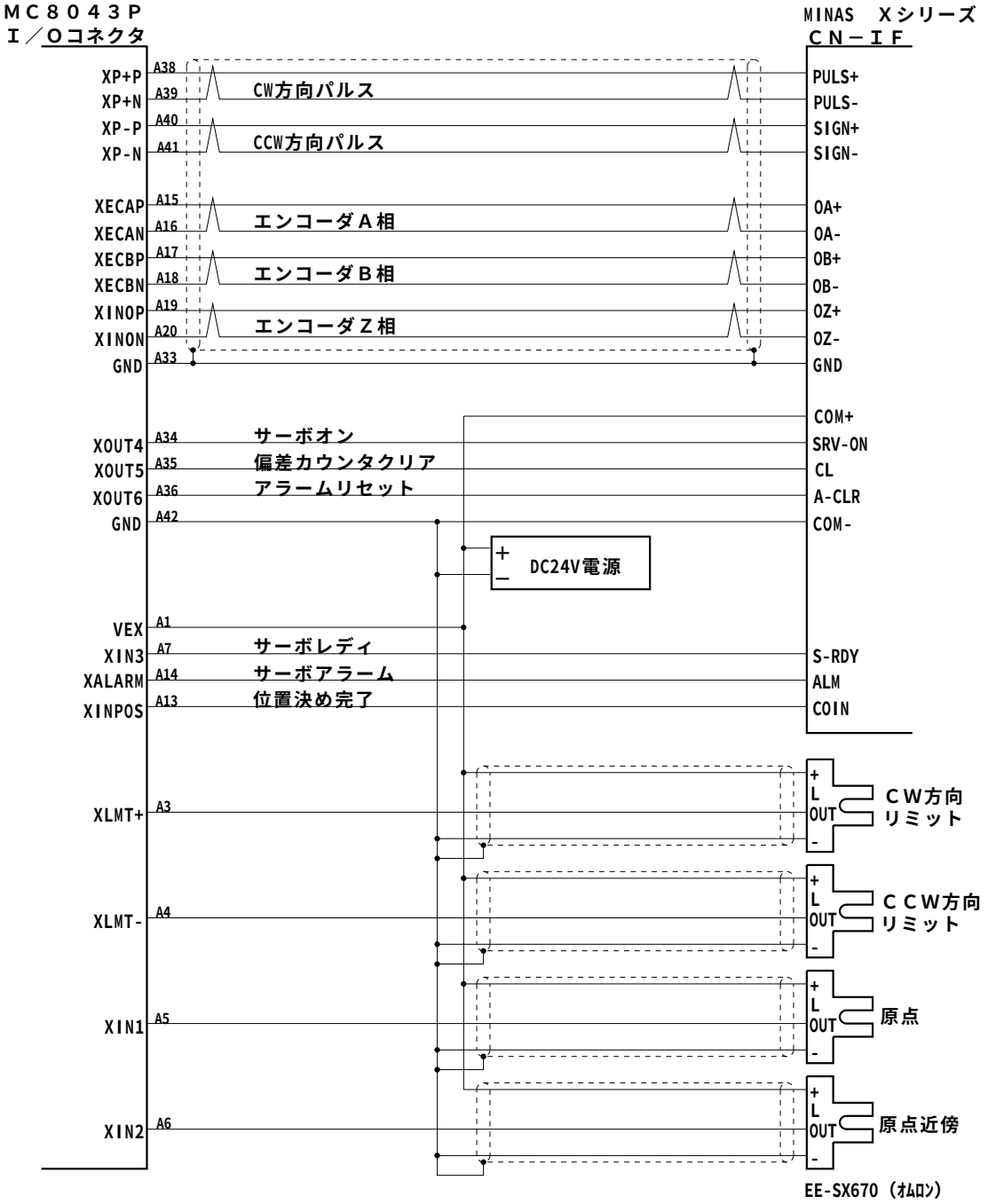


注１：ホールドＯＦＦ、励磁タイミング、オーバーヒート信号は必要に応じて配線します。ホールドＯＦＦ信号は、MCX314AsのWR3レジスタのD8ビットに0,1を書き込むことによって制御します。励磁タイミング信号は、WR1レジスタのD0,1ビットをモード設定して、原点検出動作を行わせることができます。オーバーヒート信号は、WR2レジスタのD12,13ビットをモード設定してアラーム機能を働かせることができます。また、励磁タイミング、オーバーヒート信号は、RR4,5レジスタを通して直接信号レベルを読み出すことができます。

注２：強いノイズ環境下、あるいはドライバまでの距離が長い場合は、上図のようにツイストペアシールド線を推奨します。

5.2　ＡＣサーボモータドライバとの接続例

下図は、MC8043PのＸ軸とMINAS Ｘシリーズ ＡＣサーボモータドライバとの接続例を示しています。



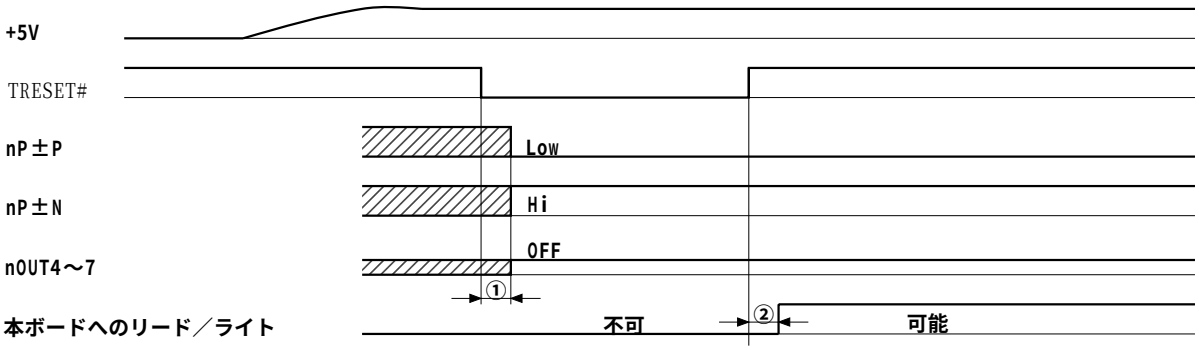
注１：ドライバの制御モード設定は位置制御モードに、指令パルス形態はＣＷ／ＣＣＷパルスモードにパラメータセットします。
　　指令パルス形態をパルス／符号モードにすると、t6時間が不足しますので適当ではありません。

注２：エンコーダＡ／Ｂ相信号はMCX314As内で実位置カウンタをカウントさせる場合に接続します。ＣＰＵ側で実位置データを
　　必要としなければ接続する必要はありません。その他の信号も必要に応じて接続します。

注３：強いノイズ環境下、あるいはドライバまでの距離が長い場合は、上図のようにツイストペアシールド線を推奨します。

６．入出力信号タイミング

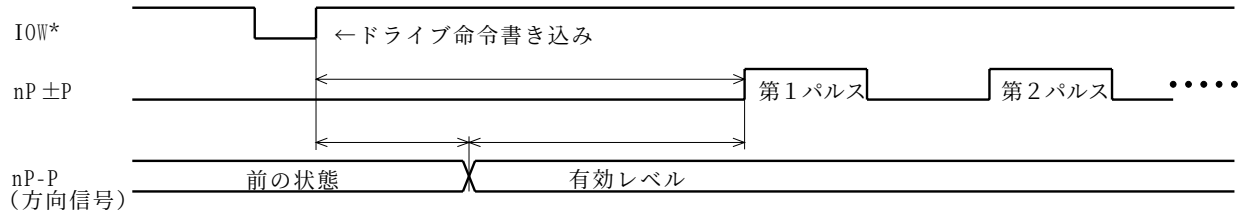
6.1 リセット時



① ドライブパルス出力信号（nP±P,nP±N）、および汎用出力信号（nOUT4～7）は、APIC21(アドテック社)のターゲットリセット信号（TRESET#）の↓から最大２５０ｎSEC以内に確定します。

② 本ボードへの書き込み／読み出しは、ターゲットリセット信号（TRESET#）の↑から５００ｎSEC後から可能になります。

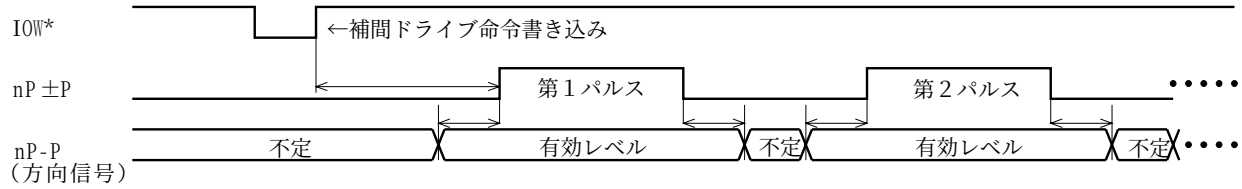
6.2 独立ドライブ開始時



ドライブ命令が書き込まれてから最大６５０nSEC以内に第１ドライブパルスが出力されます。

ドライブ出力パルス方式を１パルス方式にしたときは、ドライブ命令書き込み後最大２７５nSEC以内に方向信号（nP-P）が有効レベルになり、方向信号が有効レベルになってから３７５nSEC後に第１ドライブパルスが出力されます。

6.3 補間ドライブ時

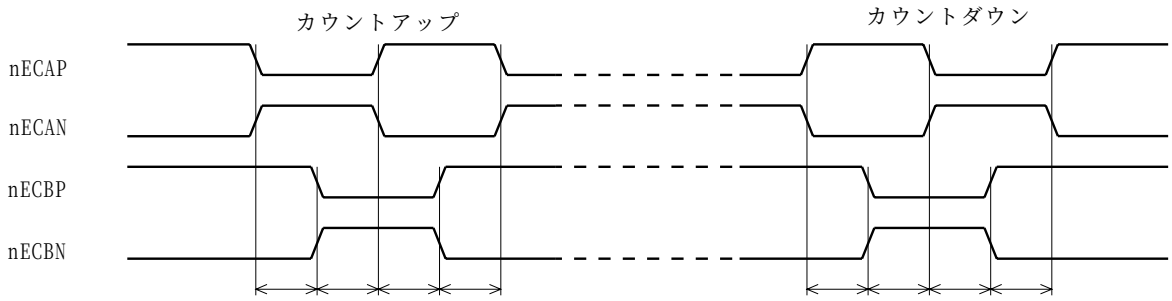


補間ドライブ時は、補間ドライブ命令が書き込まれてから最大７７５nSEC以内に第１ドライブパルスが出力されます。

ドライブ出力パルス方式を１パルス方式にしたときは、方向信号（nP-P）は各ドライブパルスHiレベル期間とその前後１２５nSECの間だけ有効レベルとなります。（ドライブパルス：正論理パルスのとき）

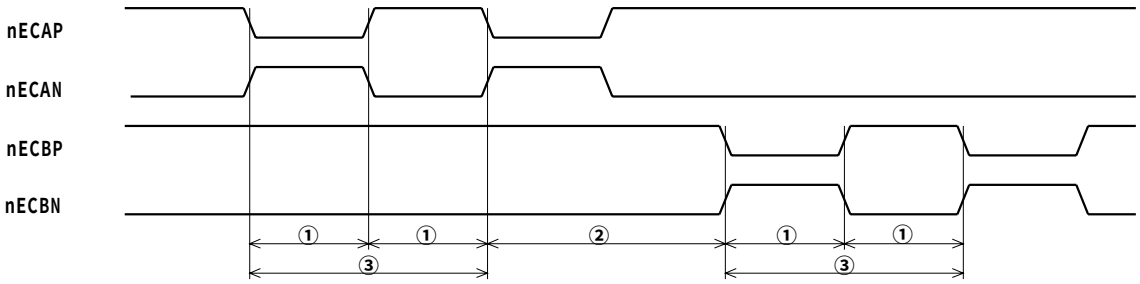
6.4　入力パルスタイミング

■ エンコーダ２相パルス入力時



（EC-A,EC-B位相差時間）　：最小２００ｎSEC

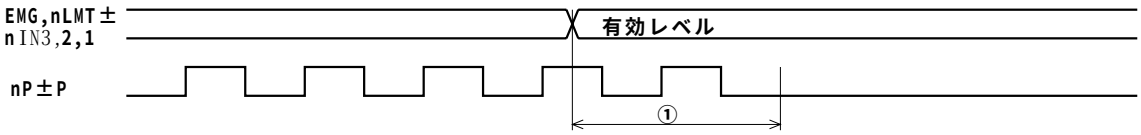
■ アップ/ダウンパルス入力時



①（UP/DOWNパルス幅）　：最小１３０ｎSEC　　**②（UP⇔DOWNパルス間）：最小２６０ｎSEC**
③（UP/DOWNパルス周期）：最小２６０ｎSEC

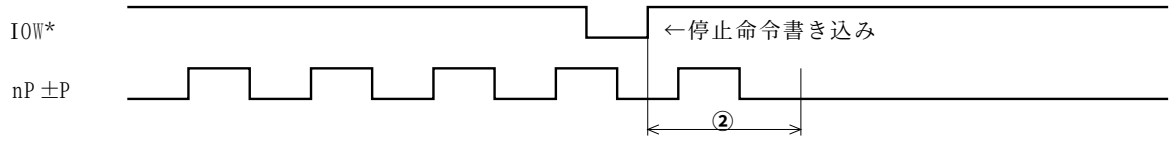
6.5　即停止タイミング

■ 外部信号による即停止



①　ドライブ途中で即停止信号が有効レベルになると、フォトカプラ遅延時間（最大100μsec）＋ＩＣ内部積分フィルタの遅延時間（初期値512μsec）＋１ドライブパルス後に停止します。

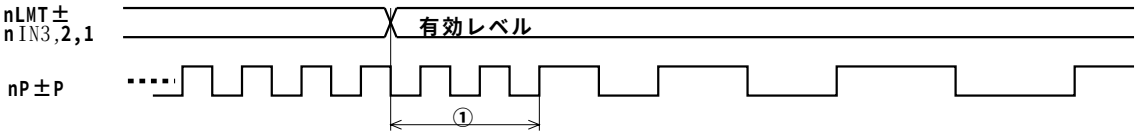
■ 命令による即停止



②　ドライブ途中で停止命令が書き込まれると、最大１ドライブパルス後に停止します。

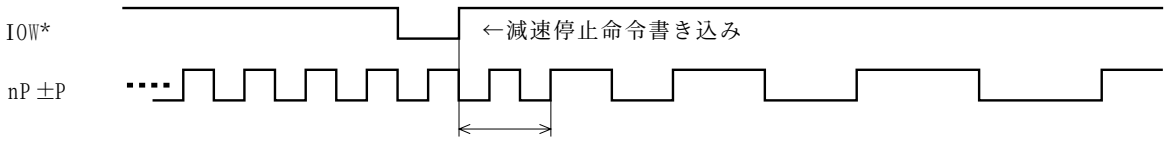
6.6　減速停止タイミング

■ 外部信号による減速停止



① ドライブ途中で外部減速停止信号が有効レベルになると、フォトカプラ遅延時間（最大100μsec）＋ＩＣ内蔵積分フィルタの遅延時間（初期値512μsec）＋２ドライブパルス後に減速を開始します。

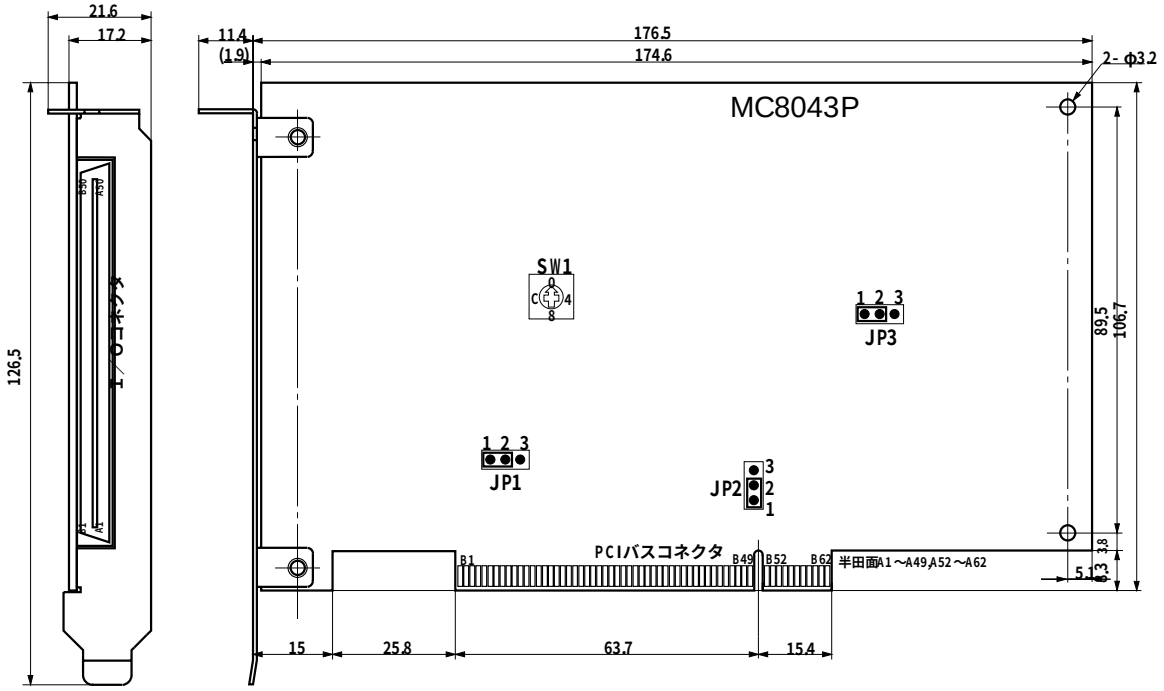
■ 命令による減速速止



ドライブ途中で減速停止命令が書き込まれると、最大２ドライブパルス後に減速を開始します。

7．基板外形

単位：mm



JP1:　緊急停止信号(EMG)のアクティブ論理を選択します。
1-2短絡(出荷時)　：信号をGNDと短絡するとアクティブになります。
2-3短絡　　：信号オープンでアクティブになります。

JP2:　1-2短絡（出荷時の状態）のままにしておいてください。

JP3:　nIN0/nIN2信号の切替えを行ないます。
1-2短絡(出荷時)　：基板I/O コネクタnIN0P/N信号がMCX314AsのnIN0に接続され、
基板I/O コネクタnIN2信号がMCX314AsのnIN2に接続されます。
2-3短絡　　：基板I/O コネクタnIN0P/N信号がMCX314AsのnIN2に接続され、
基板I/O コネクタnIN2信号がMCX314AsのnIN0に接続されます。

SW1:　ボードを複数枚使用するときのボード番号を設定するロータリスイッチです。
０～９の値を設定することができます。（出荷時：０）

８．インストール

この章では、本ボードのパソコンへの組込みとドライバのインストール方法について説明します。

8.1 ドライバソフトウェアの準備

ＣＤ－ＲＯＭからドライバをインストールする場合は、MC8043P のＣＤ－ＲＯＭを用意して下さい。
ホームページからダウンロードしたドライバをインストールする場合は、ダウンロードしたソフトウェアを解凍して下さい。

8.2 パソコンへの本ボードの組込み

① パソコン本体の電源がＯＦＦであることを確認してから、外装カバー､スロットカバー等を外します。
② 空いている拡張スロットへ本製品を差し込みます｡基板のエッジコネクタをパソコンのＰＣＩバスコネクタに正しく挿入してください。
③ 取付金具をネジ止めしてください。この時キチンとねじを締めないと後で抜け落ちたりするなどして､ショートや故障､誤動作の原因となります。
④ パソコン本体の外装カバーを元通りに取り付けます。

注意：
パソコンへの取り付け作業は必ずパソコンの電源を切断してから行ってください。さもないと回路素子を破壊する原因となります。

＜本ボードを複数枚使用する場合＞
本ボードを１つのシステム（ＰＣ）で複数枚ご使用になる時は、ＰＣＩバス上でそれぞれのボードを個別に認識させる為に、２枚以降のボードはボード番号をボード上のロータリスイッチで設定しなければなりません。ロータリスイッチ（ＳＷ１）の位置は、７章“基板外形”を参照してください。

8.3 デバイスドライバのインストール

本デバイスドライバは、「9.1.1　動作環境」で対応する全ての言語に共通です。
本デバイスドライバは本ボードを同時に１０枚まで認識します。

各ＯＳに対応したインストール手順を説明します。

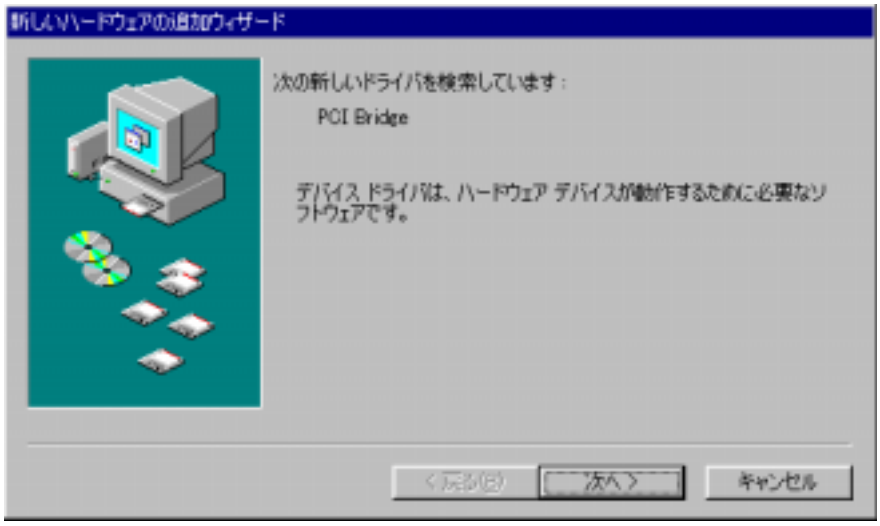
8.3.1 Windows98

①　8.1の方法でインストールするデバイスドライバを準備して下さい。

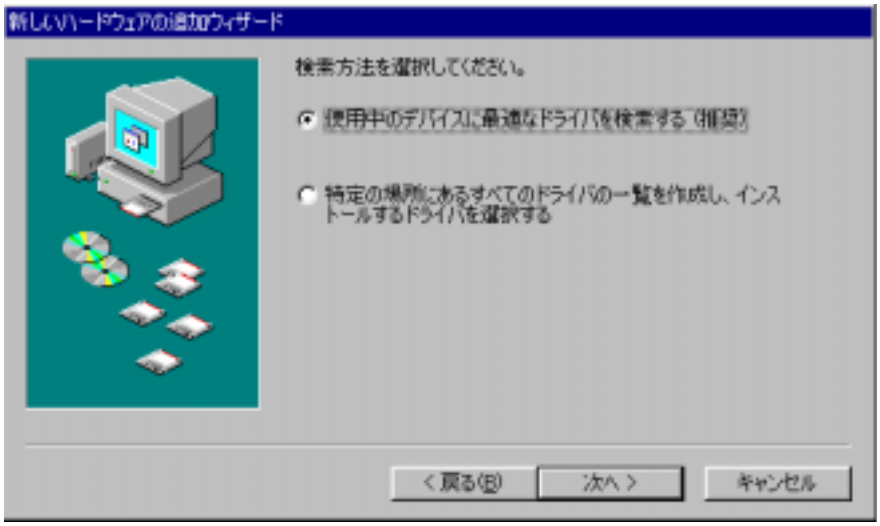
②　8.2によって本ボードが確実にパソコンに組み込まれているか確認してください。

③　パソコン本体の電源をＯＮし、Windows98を起動します。

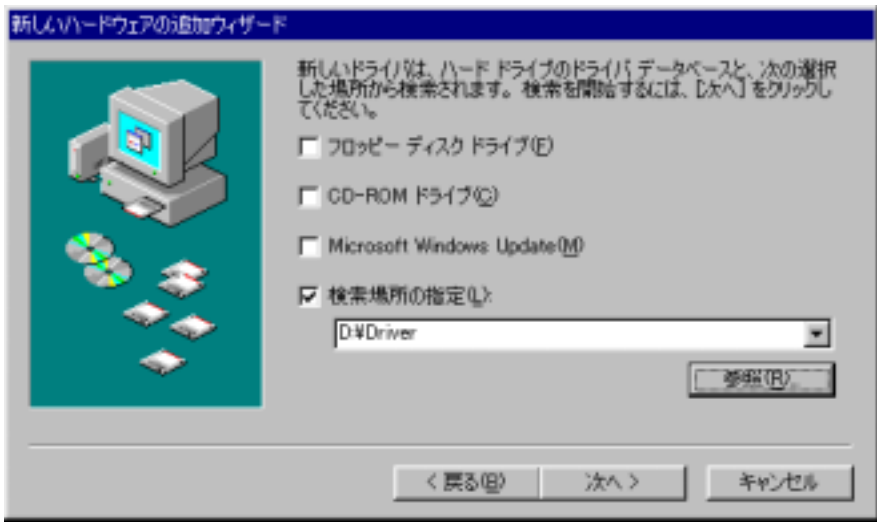
④　すぐに「新しいハードウェア」と表示され「ハードウェア追加ウィザード」が起動しますので、[次へ]をクリックします。



⑤　デバイスドライバの検出方法を選択する画面が表示されますので、「使用中のデバイスに最適なドライバを検索する(推奨)」にチェックをして[次へ]をクリックします。



⑥　検索場所の指定画面が表示されますので[検索場所の指定]をチェックします。
ドライバをＣＤ－ＲＯＭからインストールする場合は、パソコンにＣＤ－ＲＯＭをセットし、ＣＤ－ＲＯＭが認識されてから次の動作に進みます。
参照ボタンを押し、提供ＣＤ－ＲＯＭのDriverフォルダ（提供ＣＤ－ＲＯＭがＤドライブにある場合は、D:\Driver）、あるいはダウンロードしたソフトウェアのDriverフォルダを選択し、テキストボックスにフォルダが表示されたら、[次へ]をクリックします。



⑦　[MC8043P Device]が検出できたら[次へ]をクリックします。

⑧　デバイスドライバのコピーが完了すると、メッセージが表示されますので[完了]をクリックします。

⑨　以上でデバイスドライバのインストールは完了です。
次の方法で正しくインストールされたかどうかを確認して下さい。
「コントロールパネル」－「システム」－「デバイスマネージャ」画面（下記左画面）を開き、
「ＮＯＶＡ」の下の「ＭＣ８０４３Ｐ　Ｄｅｖｉｃｅ」をダブルクリックし、「全般」タブで下記右画面を表示します。
この画面でデバイスの状態に「このデバイスは正常に動作しています」と記載されていたらインストールは正常終了です。

また、デバイスドライバのインストール完了後はパソコンの起動時に手順④のようにハードウェアウィザードが起動することはありません。もしハードウェアウィザードが起動するような場合はインストールが正常に終了していない可能性がありますので、その場合は8.4の手順に従って一度本ボードを取り外した後に8.2の手順から再度インストールをやり直してください。

インストールを完了したらリソース(Ｉ／Ｏアドレス、割り込みレベル)の設定､競合の有無を[コントロールパネル]-[システム]-[デバイスマネージャ]タブで確認してくだい。

8.3.2 Windows2000

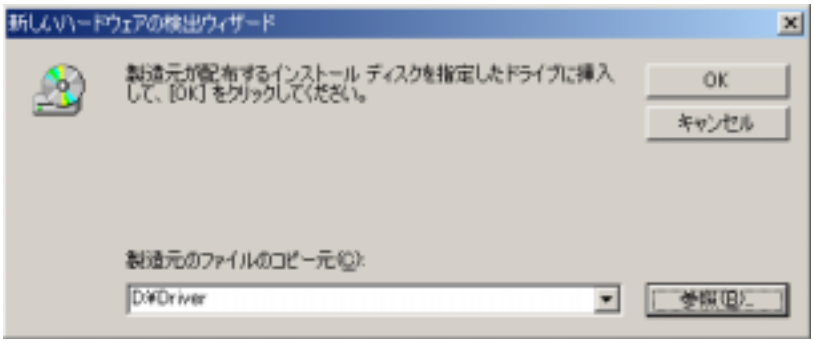
デバイスドライバのインストールは必ずアドミニストレーター権限をもったユーザーログインで行ってください。アドミニストレーター権限以外でインストールをした場合正常にインストールされません。

① 8.1の方法でインストールするデバイスドライバを準備して下さい。
② 8.2によって本ボードが確実にパソコンに組み込まれているか確認してください。
③ パソコン本体の電源をＯＮし、Windows2000を起動します。
④ アドミニストレーター権限を持ったユーザーでログインしてください。
⑤ すぐに「新しいハードウェアが見つかりました」と表示されデバイスドライバのインストールが開始されます。
⑥ すぐに「新しいハードウェアの検出ウィザード」が表示され「新しいハードウェアの検索ウィザードの開始」が出ますので[次へ]をクリックします。

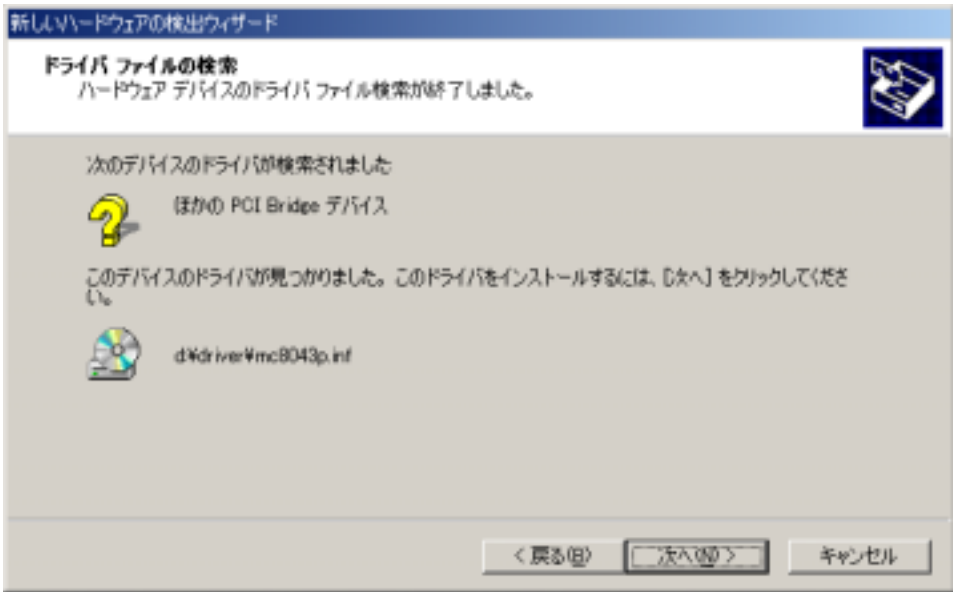
⑦ 「ハードウェアデバイスドライバのインストール」が表示され、デバイスドライバの自動検出を促す画面が出ますので、「デバイスに最適なドライバを検出する(推奨)」にチェックして[次へ]をクリックします。

⑧　検索場所の指定が出ますので[場所を指定]にチェックして[次へ]をクリックします。

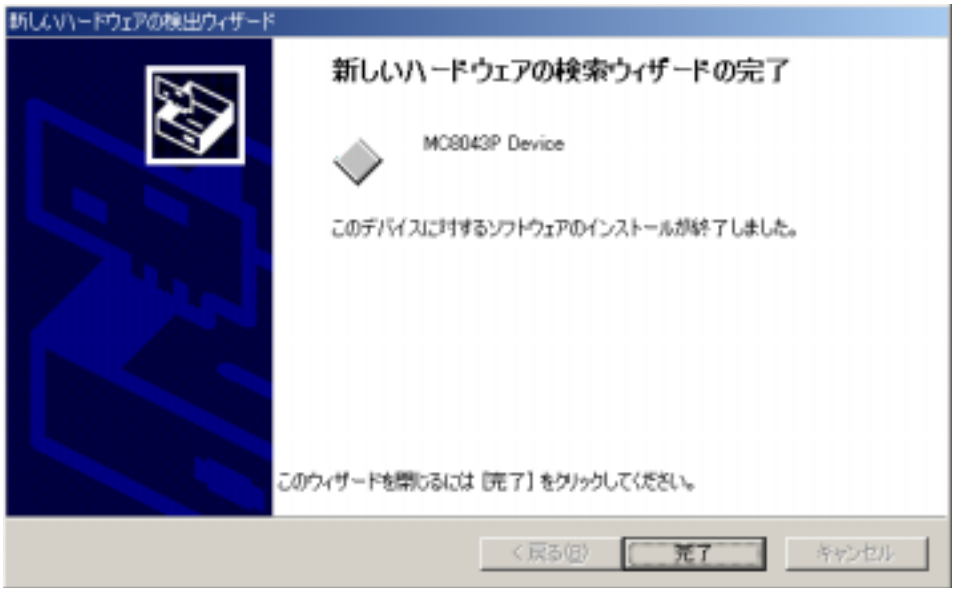
⑨　下のような画面が出ます。
ドライバをＣＤ－ＲＯＭからインストールする場合は、パソコンにＣＤ－ＲＯＭをセットし、ＣＤ－ＲＯＭが認識されてから次の動作に進みます。
参照ボタンを押し、提供ＣＤ－ＲＯＭのDriverフォルダ（提供ＣＤ－ＲＯＭがＤドライブにある場合は、D:\Driver）、あるいはダウンロードしたソフトウェアのDriverフォルダを選択し、テキストボックスにフォルダが表示されたら、ＯＫを押してください。



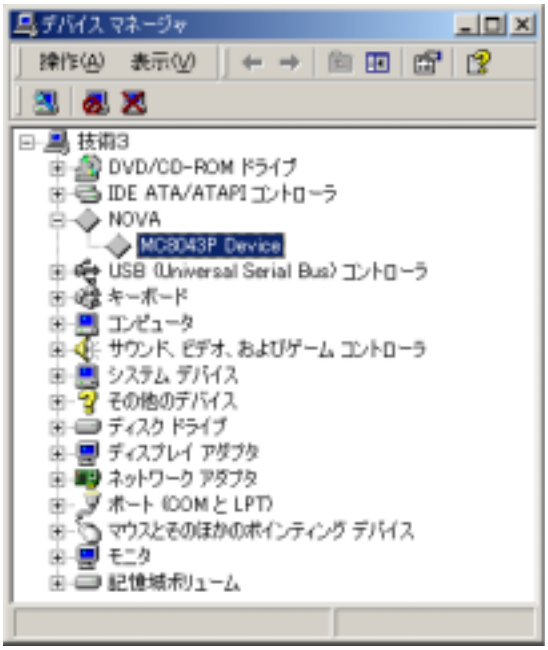
⑩　デバイスドライバの情報ファイルが発見されると確認の画面が出ますので、「\ driver\ mc8043p.inf」が表示されたことを確認して、[次へ]をクリックします。

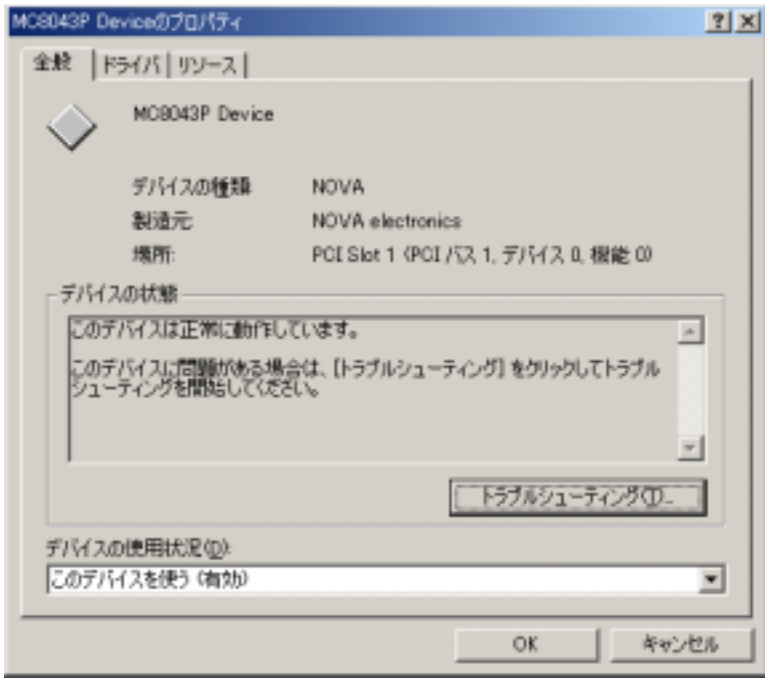


⑪　デバイスドライバのインストールが正常に完了すると、メッセージが出ますので、[完了]をクリックします。



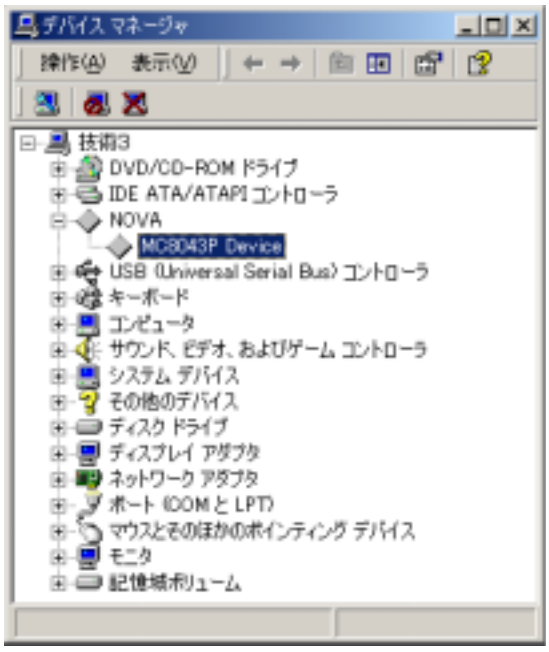
⑫　以上でデバイスドライバのインストールは完了です。
次の方法で正しくインストールされたかどうかを確認して下さい。
「コントロールパネル」－「システム」－「ハードウェア」－「デバイスマネージャ」画面（下記左画面）を開き、
「ＮＯＶＡ」の下の「ＭＣ８０４３Ｐ　Ｄｅｖｉｃｅ」をダブルクリックし、「全般」タブで下記右画面を表示します。
この画面でデバイスの状態に「このデバイスは正常に動作しています」と記載されていたらインストールは正常終了です。

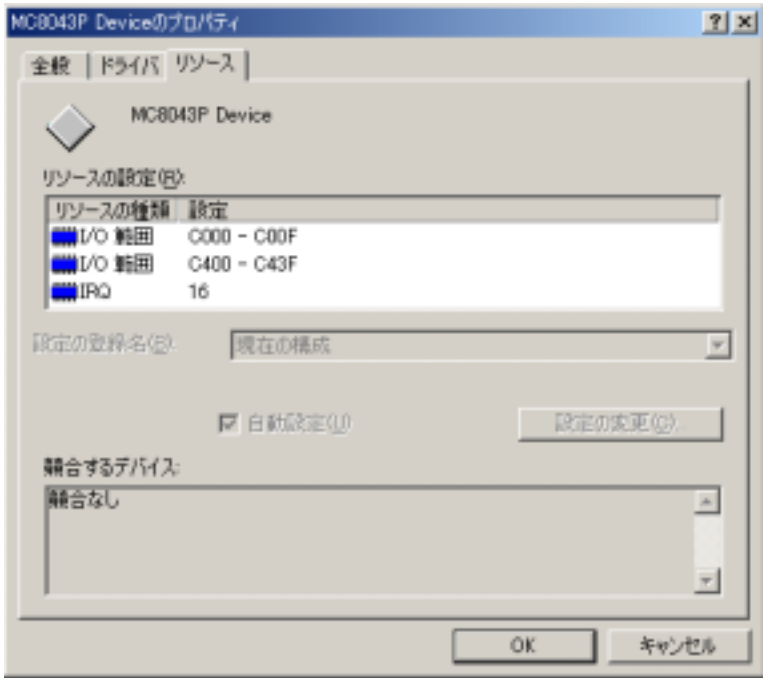




また、デバイスドライバのインストール完了後はパソコンの起動時に手順⑥のようにハードウェアウィザードが起動することはありません。もしハードウェアウィザードが起動するような場合はインストールが正常に終了していない可能性がありますので、その場合は8.4の手順に従って一度本ボードを取り外した後に8.2の手順から再度インストールをやり直してください。

インストールを完了したらリソース(Ｉ／Ｏアドレス、割り込みレベル)の設定､競合の有無を[コントロールパネル]-[システム]-[ハードウェア]-[デバイスマネージャ]の[プロパティ]で確認してください。



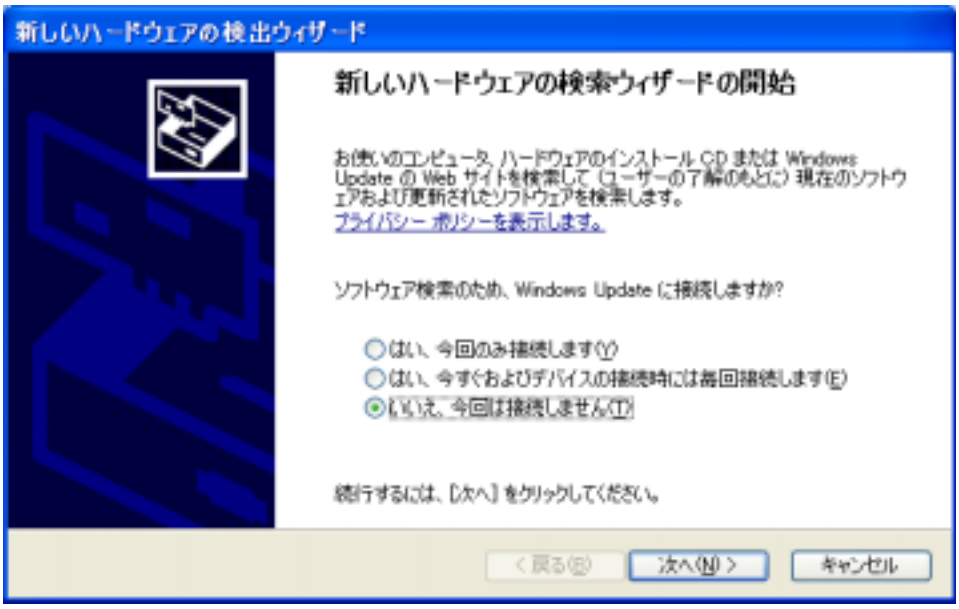


8.3.3 WindowsXP

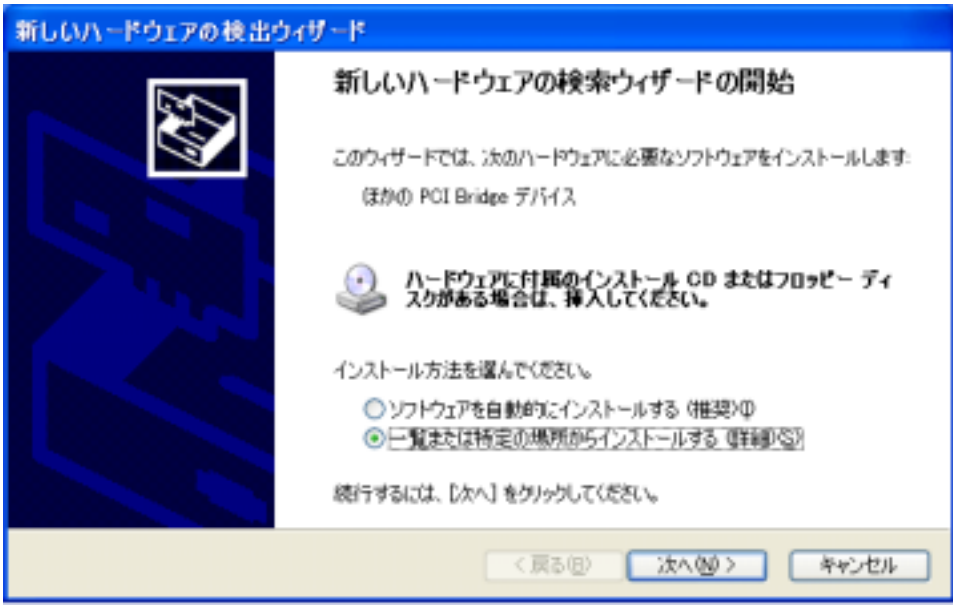
デバイスドライバのインストールは必ずアドミニストレーター権限をもったユーザーログインで行ってください。アドミニストレーター権限以外でインストールをした場合正常にインストールされません。

① 8.1の方法でインストールするデバイスドライバを準備して下さい。
② 8.2によって本ボードが確実にパソコンに組み込まれているか確認してください。
③ パソコン本体の電源をＯＮし、WindowsXPを起動します。
④ アドミニストレーター権限を持ったユーザーでログインしてください。

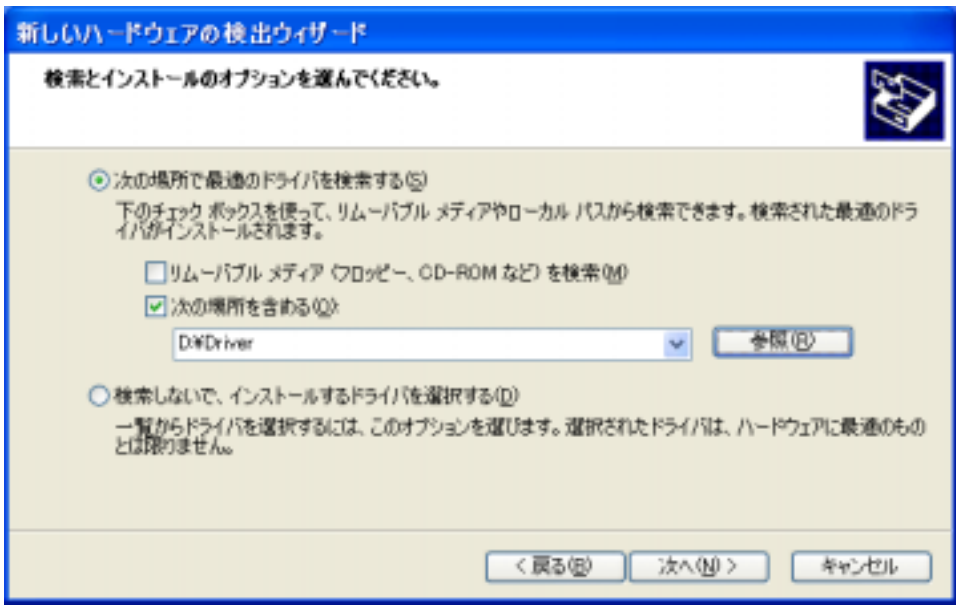
⑤ WindowsXP サービスパック２の場合は、下の画面が表示されますので、「いいえ、今回は接続しません」を選択し、[次へ]をクリックします。
WindowsXP サービスパック１の場合は、この画面は表示されませんので、次に進んで下さい。



⑥ 次に下の画面が表示されますので「一覧または特定の場所からインストールする」を選択し、[次へ]をクリックします。

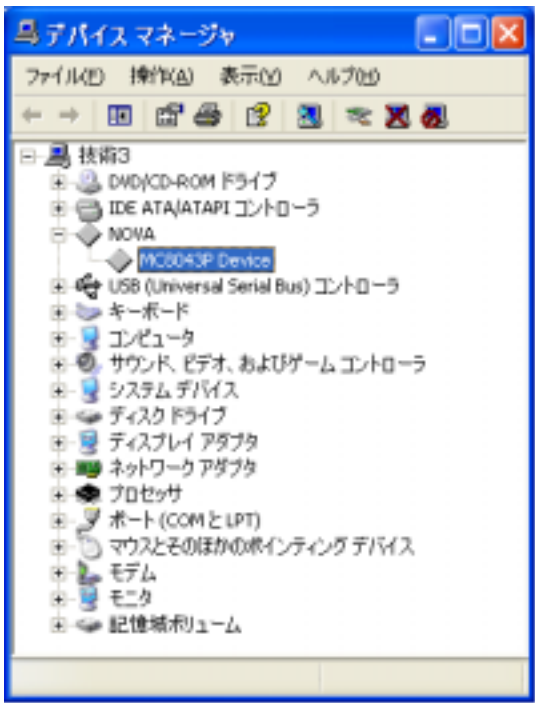


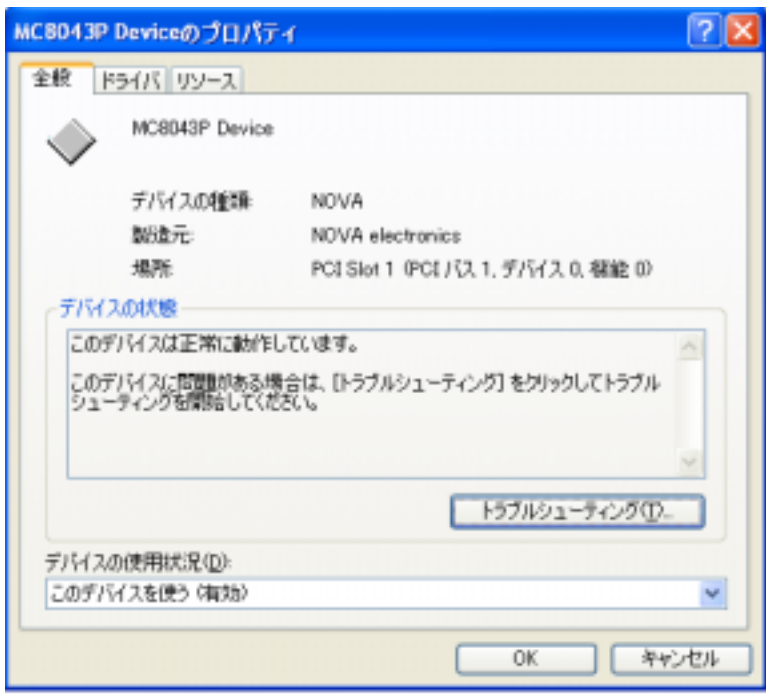
⑦　下の画面が表示されますので、「次の場所で最適のドライバを検索する」を選択し、「次の場所を含める」をチェックします。
ドライバをＣＤ－ＲＯＭからインストールする場合は、パソコンにＣＤ－ＲＯＭをセットし、ＣＤ－ＲＯＭが認識されてから次の動作に進みます。
参照ボタンを押し、提供ＣＤ－ＲＯＭのDriverフォルダ（提供ＣＤ－ＲＯＭがＤドライブにある場合は、D:\Driver）、あるいはダウンロードしたソフトウェアのDriverフォルダを選択し、テキストボックスにフォルダが表示されたら、[次へ]をクリックします。



⑧　デバイスドライバのインストールが正常に完了すると、下の画面が表示されますので、[完了]をクリックします。

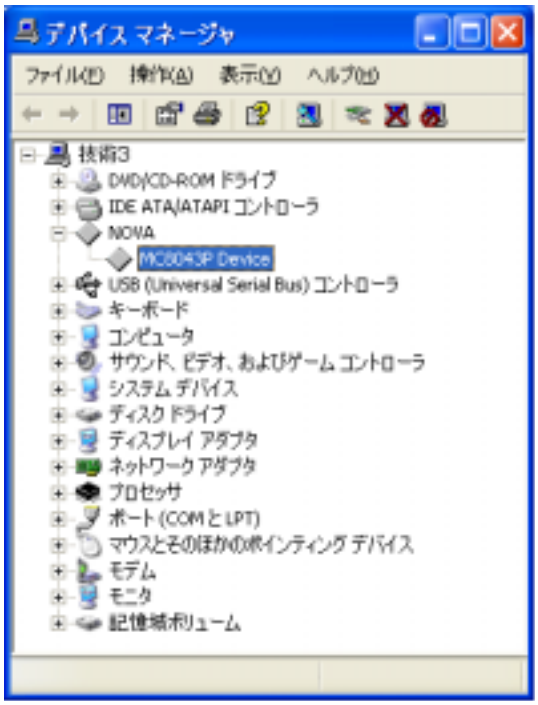
⑨　以上でデバイスドライバのインストールは完了です。
次の方法で正しくインストールされたかどうかを確認して下さい。
「コントロールパネル」－「システム」－「ハードウェア」－「デバイスマネージャ」画面（下記左画面）を開き、
「ＮＯＶＡ」の下の「ＭＣ８０４３Ｐ　Ｄｅｖｉｃｅ」をダブルクリックし、「全般」タブで下記右画面を表示します。
この画面でデバイスの状態に「このデバイスは正常に動作しています」と記載されていたらインストールは正常終了です。

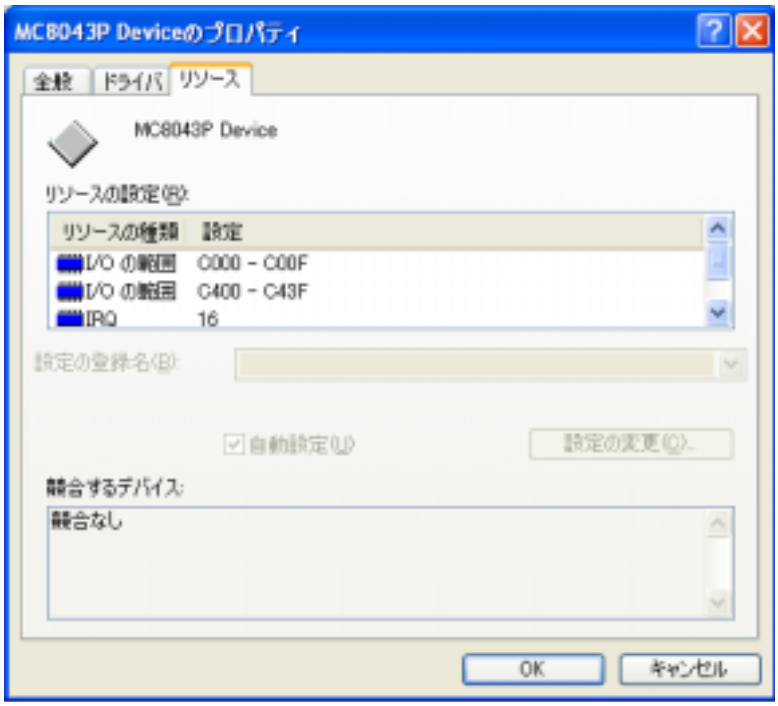




また、デバイスドライバのインストール完了後はパソコンの起動時に手順⑤、⑥のようにハードウェアウィザードが起動することはありません。もしハードウェアウィザードが起動するような場合はインストールが正常に終了していない可能性がありますので、その場合は8.4の手順に従って一度本ボードを取り外した後に8.2の手順から再度インストールをやり直してください。

インストールを完了したらリソース(Ｉ／Ｏアドレス、割り込みレベル)の設定、競合の有無を[コントロールパネル]-[システム]-[ハードウェア]-[デバイスマネージャ]の[プロパティ]で確認してください。





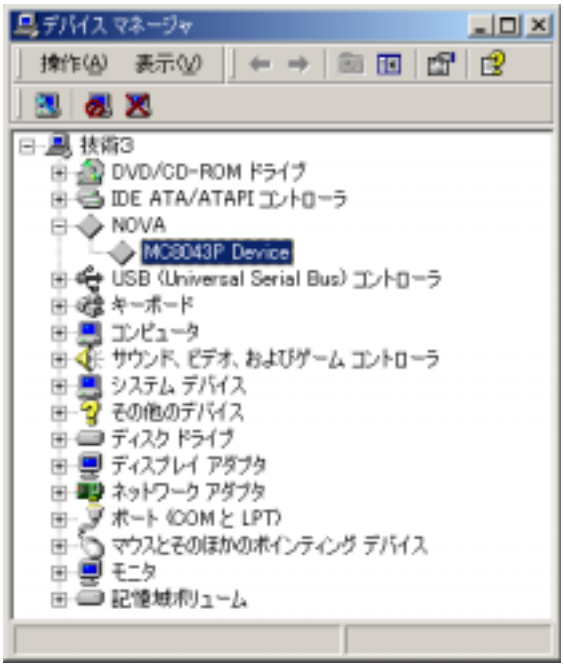
8.4 取り外し

8.4.1 Windows98

①まず[コントロールパネル]-[システム]-[デバイスマネージャ]タブで本製品のデバイスドライバを削除してください。
②パソコン本体の電源がＯＦＦであることを確認してから、外装カバー､スロットカバー等を外します。
③ボードを止めているビスを外します。
④本ボードを指先でつまんで軽く左右にゆするようにしながら引き出します。
⑤パソコン本体の電源をＯＮし、Windows98を起動します。
⑥[コントロールパネル]-[システム]-[デバイスマネージャ]タブで本製品が削除されていることを確認してください。

8.4.2 Windows2000/XP

① まず[コントロールパネル]-[システム]-[ハードウェア]-[デバイスマネージャ]で本製品のデバイスドライバを削除してください。
② パソコン本体の電源がＯＦＦであることを確認してから、外装カバー､スロットカバー等を外します。
③ ボードを止めているビスを外します。
④ 本ボードを指先でつまんで軽く左右にゆするようにしながら引き出します。
⑤ パソコン本体の電源をＯＮし、Windows2000/XPを起動します。
⑥ [コントロールパネル]-[システム]-[ハードウェア]-[デバイスマネージャ]タブで本製品が削除されていることを確認して下さい。



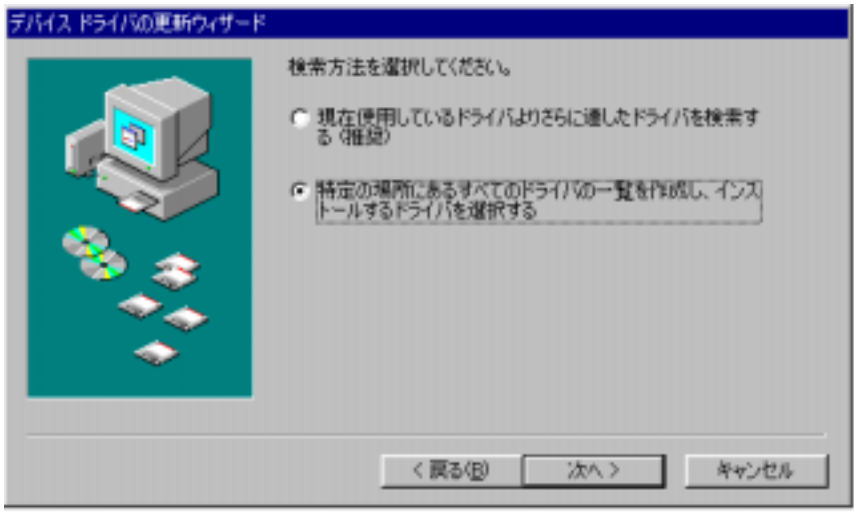
8.5　デバイスドライバの更新

デバイスドライバのバージョンが新しくなった場合、次の手順でドライバの更新を行って下さい。
以下に、各ＯＳに対応した更新手順を説明します。

8.5.1 Windows98

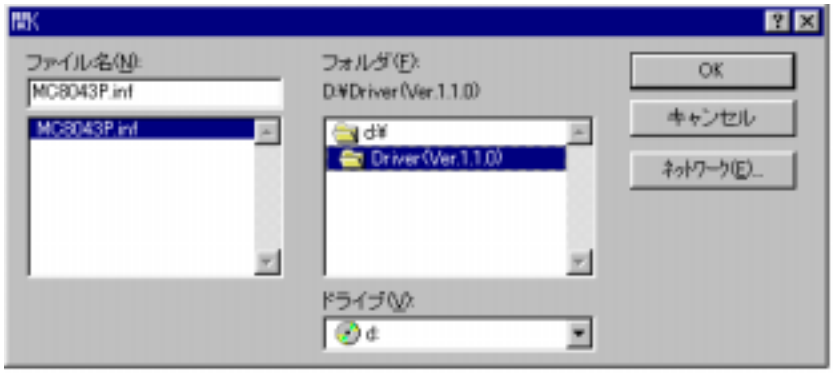
①「コントロールパネル」－「システム」－「デバイスマネージャ」画面（下記左画面）を開き、「ＮＯＶＡ」の下の「ＭＣ８０４３Ｐ　Ｄｅｖｉｃｅ」をダブルクリックし、「ドライバ」タブで下記右画面を表示します。
②次に「ドライバの更新」ボタンをクリックし、その次の画面で「次へ」ボタンをクリックします。

③下記画面が表示されますので、「特定の場所にあるすべてのドライバの一覧を作成し、インストールするドライバを選択する」を選択し、「次へ」ボタンをクリックします。



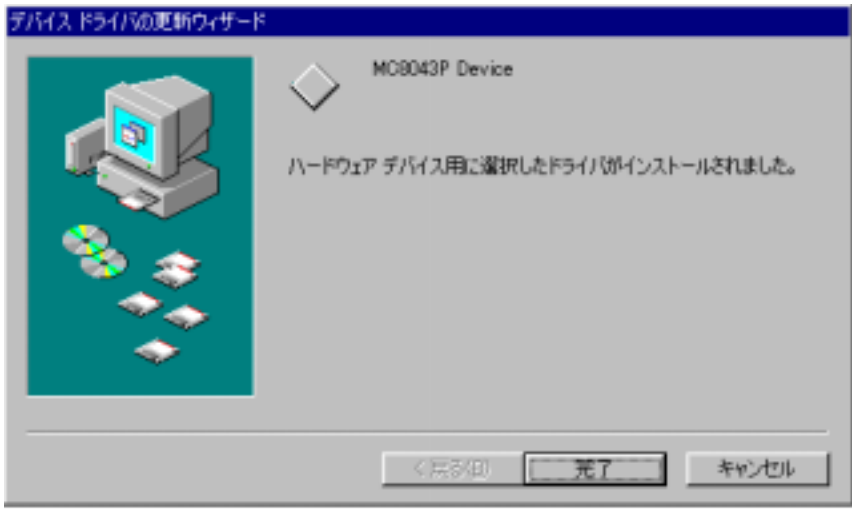
④下記画面で、「ディスク使用」ボタンをクリック後、「参照」ボタンをクリックします。

⑤下記画面で、更新するドライバソフトのフォルダ（\ Driver）を選択後、「ＯＫ」ボタンをクリックし、その次の画面でも「ＯＫ」ボタンをクリックします。



⑥下記画面で、「次へ」ボタンをクリック後、その次の画面でも「次へ」ボタンをクリックすると、ドライバが更新されます。

⑦正常に更新されると、下記画面が表示されます。



⑧下記画面（MC8043P Device のプロパティ画面）で、更新したドライバの日付を確認します。
\ Driver\ Version.txt ファイルの「１．ドライババージョン」の日付が下記画面に表示されますので正しいか確認して下さい。
次に「ドライバの詳細」ボタンをクリックします。

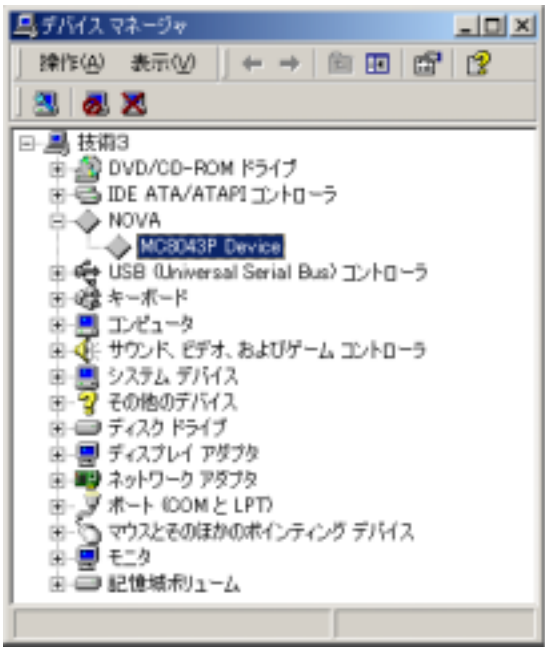
⑨下記画面で、更新したドライバのファイルバージョンを確認します。
\ Driver\ Version.txt ファイルの「２．ドライバファイルバージョン」のバージョンが下記画面に表示されます。
Version.txt ファイルに記載している２つのファイルのバージョンが正しいか確認して下さい。

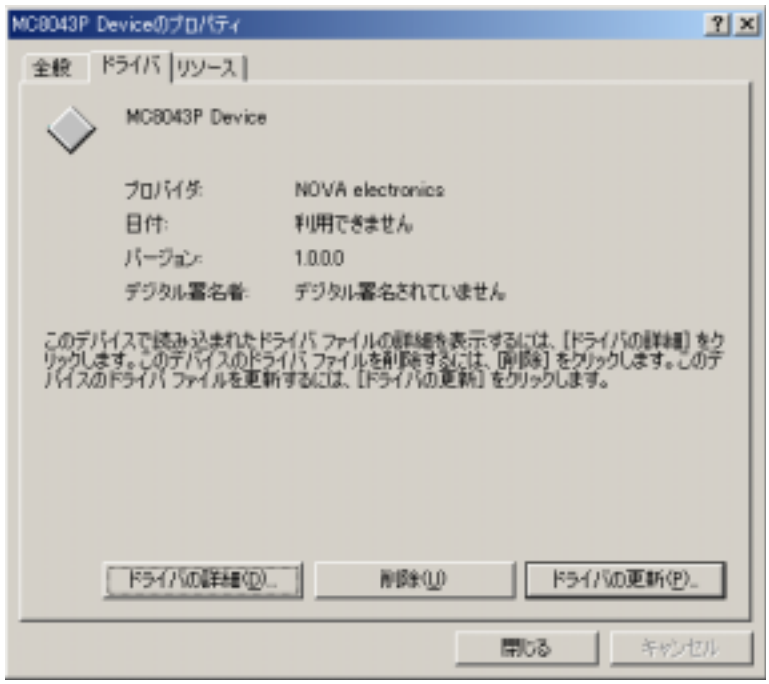
⑩ MC8043P ボードが複数ある場合は、「デバイスマネージャ」画面の「ＮＯＶＡ」の下に表示される全ての「ＭＣ８０４３Ｐ Ｄｅｖｉｃｅ」について、①からの手順でドライバを更新して下さい。

⑪パソコンを再起動して下さい。これで更新作業は終了です。

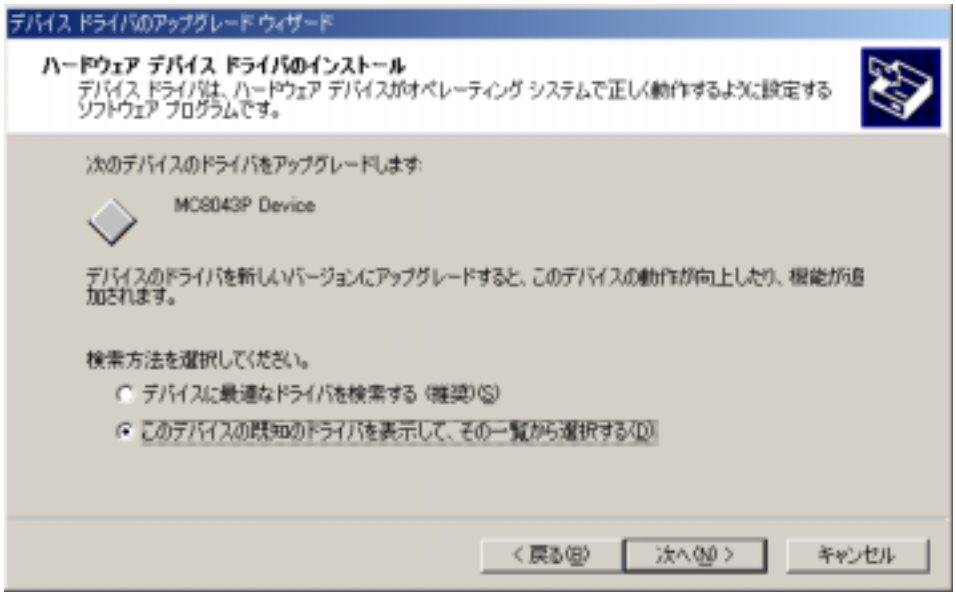
8.5.2 Windows2000

①「コントロールパネル」－「システム」－「ハードウェア」－「デバイスマネージャ」画面（下記左画面）を開き、
「ＮＯＶＡ」の下の「ＭＣ８０４３Ｐ　Ｄｅｖｉｃｅ」をダブルクリックし、「ドライバ」タブで下記右画面を表示します。
②次に「ドライバの更新」ボタンをクリックし、次に表示された画面では「次へ」ボタンをクリックします。

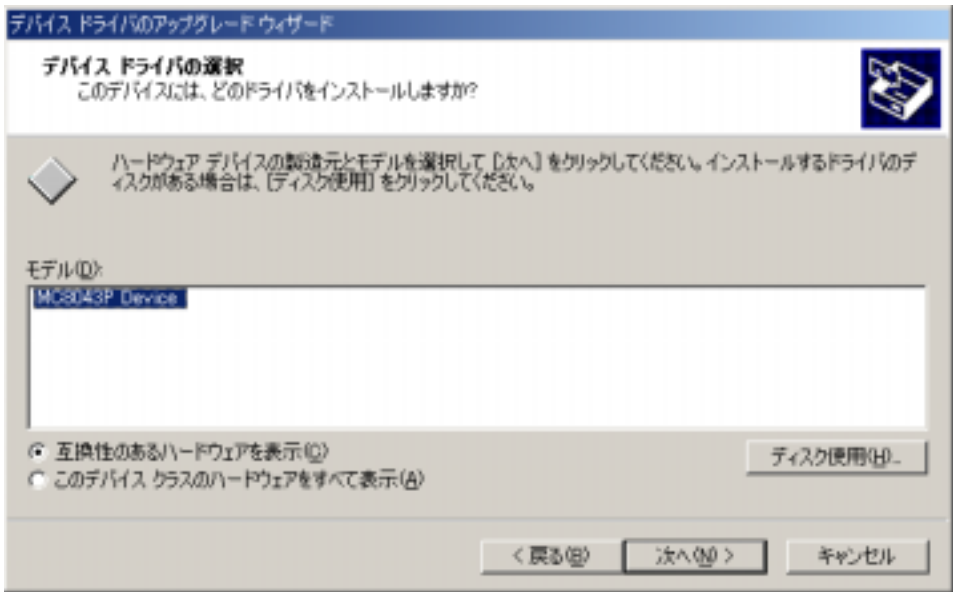




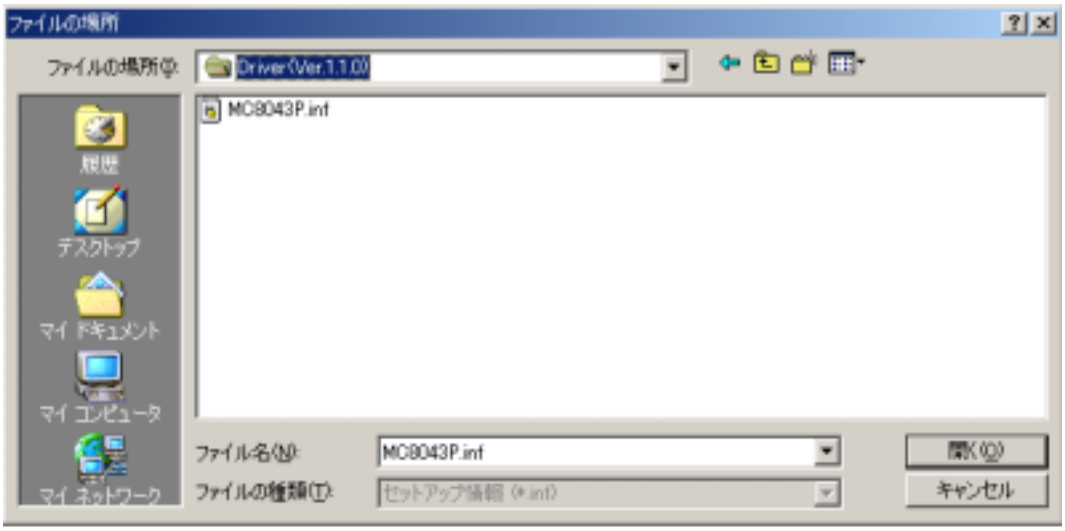
③下記画面が表示されますので、「このデバイスの既知のドライバを表示して、その一覧から選択する」を選択し、「次へ」
ボタンをクリックします。



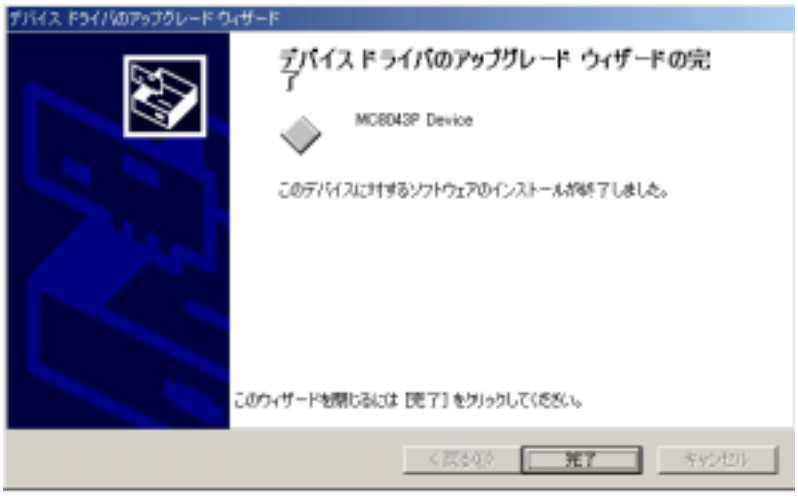
④下記画面で、「ディスク使用」ボタンをクリック後、「参照」ボタンをクリックします。



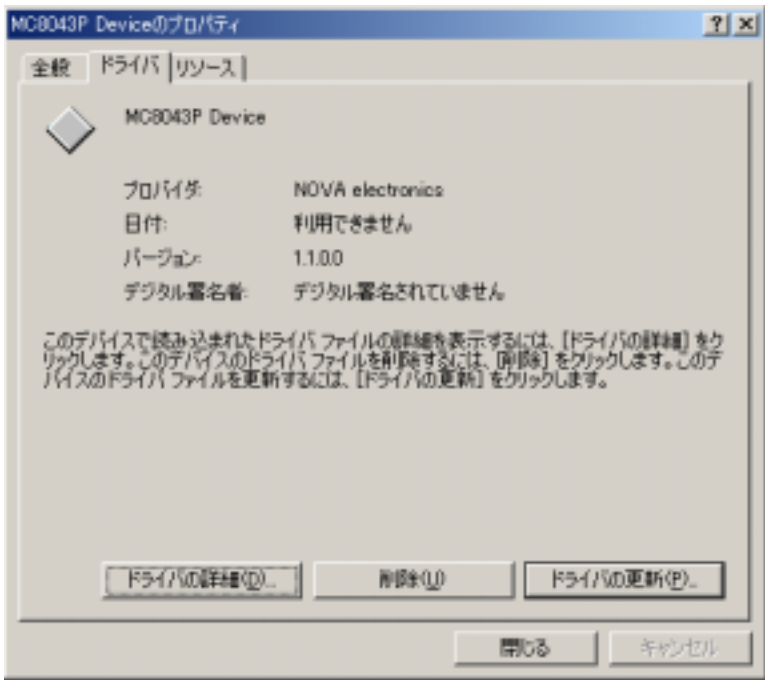
⑤下記画面で、更新するドライバソフトのフォルダ（\ Driver）を選択し、「開く」ボタンクリック後、「ＯＫ」ボタンをクリックします。その次の画面では「次へ」ボタンクリック後、更に次の画面でも「次へ」ボタンをクリックします。



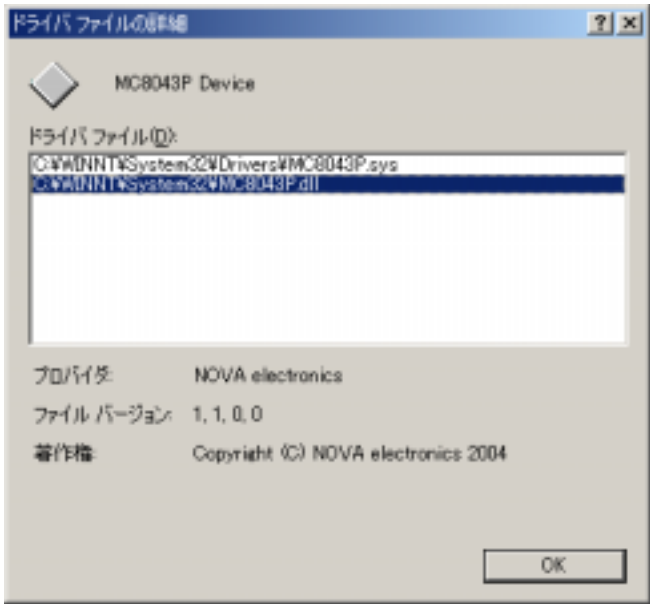
⑥正常に更新されると、下記画面が表示されます。



⑦下記画面（MC8043P Device のプロパティ画面）で、更新したドライバのバージョンを確認します。
\ Driver\ Version.txt ファイルの「１．ドライババージョン」のバージョンが下記画面に表示されますので正しいか確認して下さい。 次に「ドライバの詳細」ボタンをクリックします。



⑧下記画面で、更新したドライバのファイルバージョンを確認します。
\ Driver\ Version.txt ファイルの「２．ドライバファイルバージョン」のバージョンが下記画面に表示されます。
Version.txt ファイルに記載している２つのファイルのバージョンが正しいか確認して下さい。

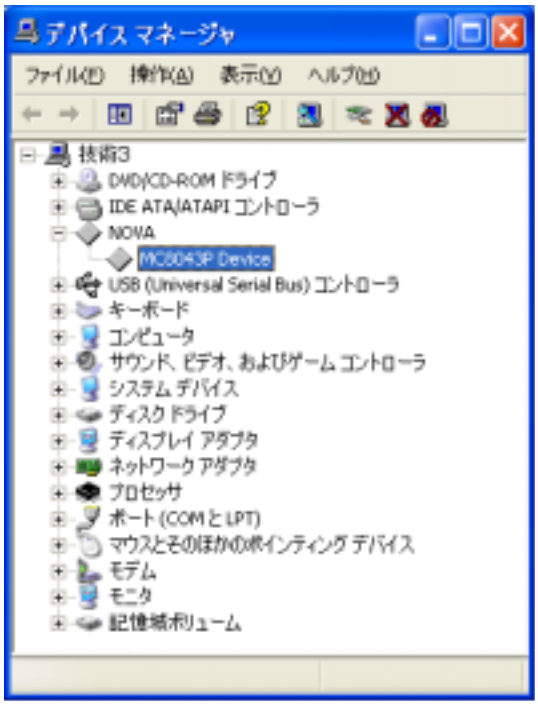


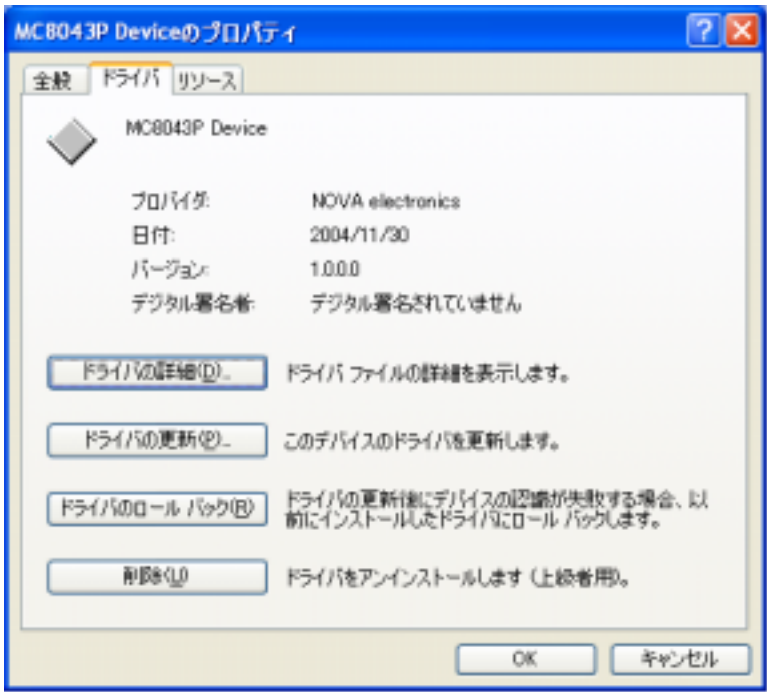
⑨MC8043P ボードが複数ある場合は、「デバイスマネージャ」画面の「ＮＯＶＡ」の下に表示される全ての「ＭＣ８０４３Ｐ Ｄｅｖｉｃｅ」について、①からの手順でドライバを更新して下さい。

⑩パソコンを再起動して下さい。これで更新作業は終了です。

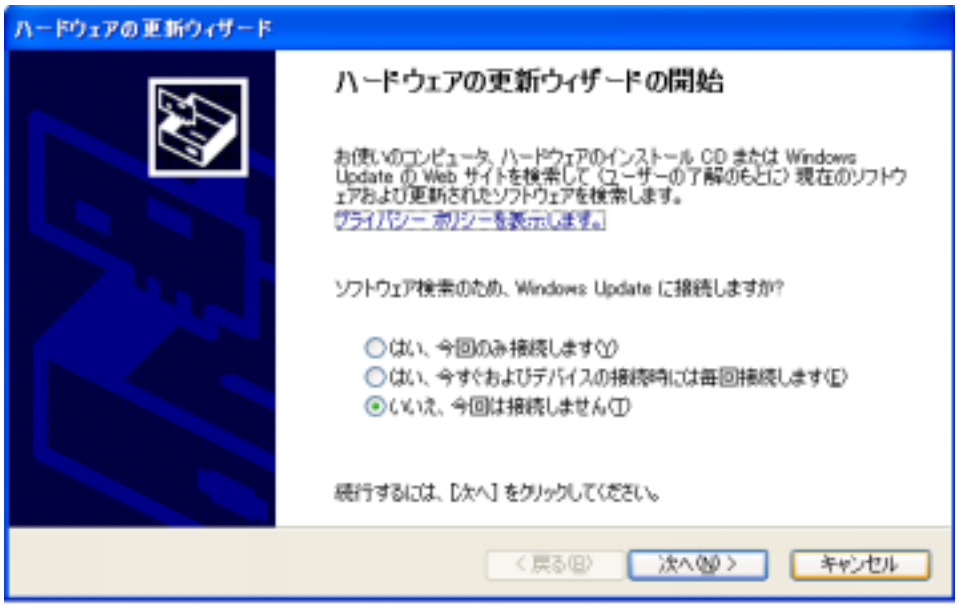
8.5.3 WindowsXP

①「コントロールパネル」－「システム」－「ハードウェア」－「デバイスマネージャ」画面（下記左画面）を開き、
「ＮＯＶＡ」の下の「ＭＣ８０４３Ｐ　Ｄｅｖｉｃｅ」をダブルクリックし、「ドライバ」タブで下記右画面を表示します。
②次に「ドライバの更新」ボタンをクリックします。

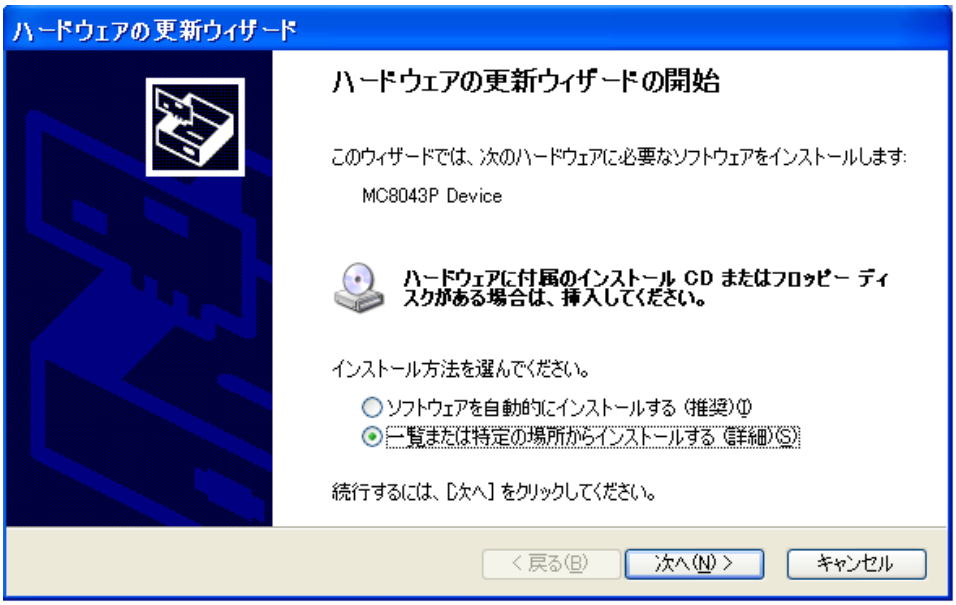




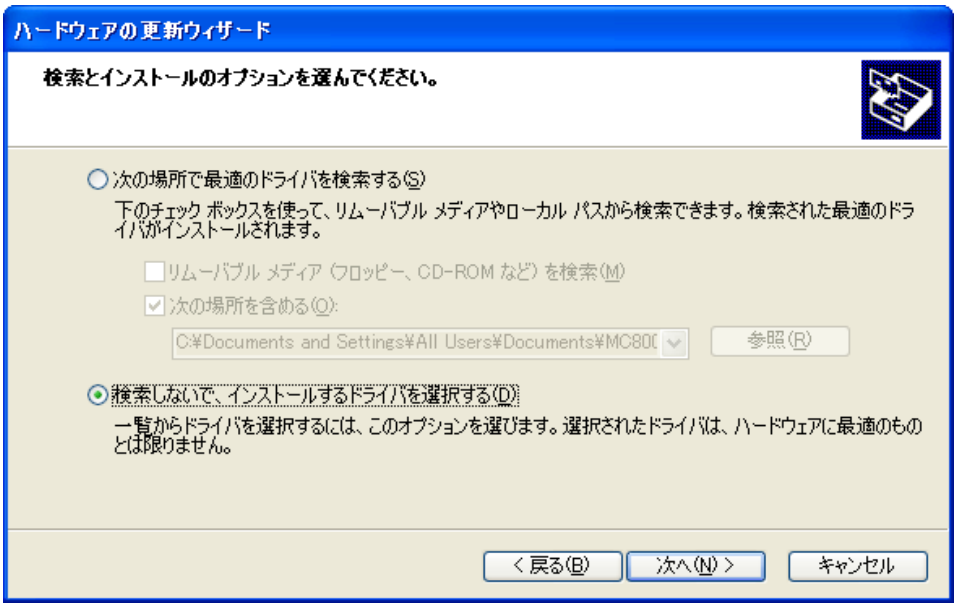
③WindowsXP サービスパック２の場合は、下の画面が表示されますので、「いいえ、今回は接続しません」を選択し、
[次へ]をクリックします。
WindowsXP サービスパック１の場合は、この画面は表示されませんので、次に進んで下さい。



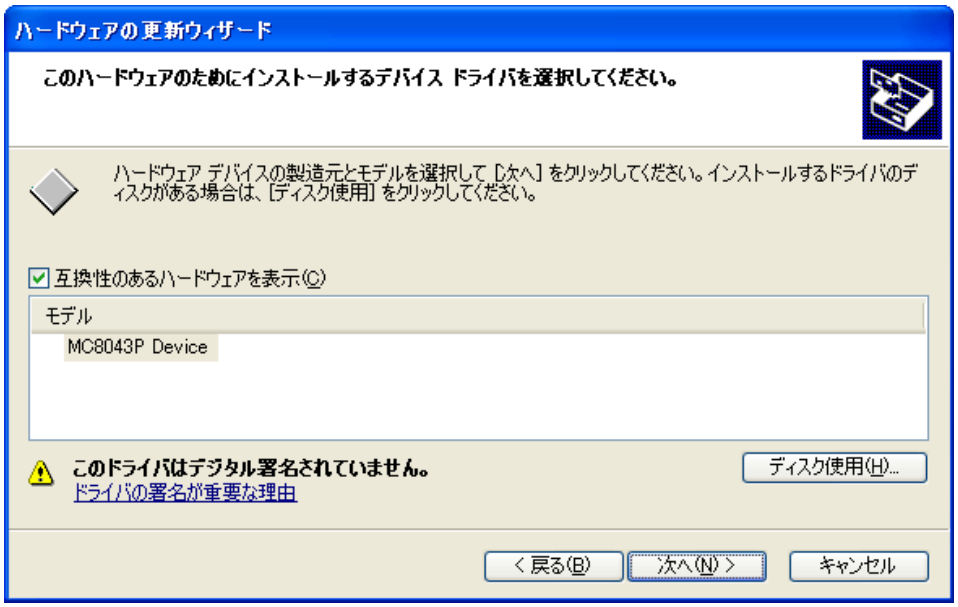
④下記画面で、「一覧または特定の場所からインストールする」を選択し、「次へ」ボタンをクリックします。



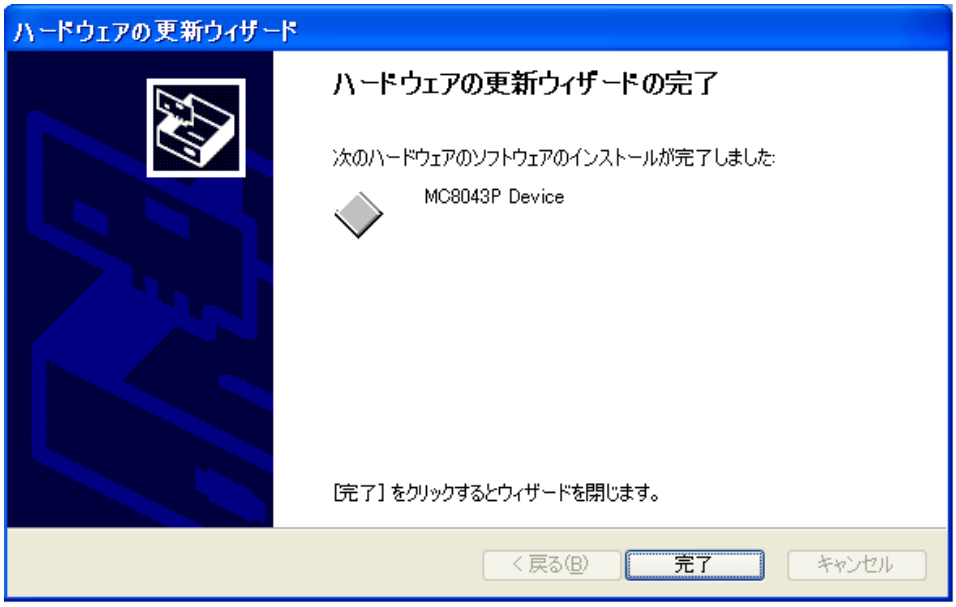
⑤下記画面で、「検索しないで、インストールするドライバを選択する」を選択し、[次へ]をクリックします。



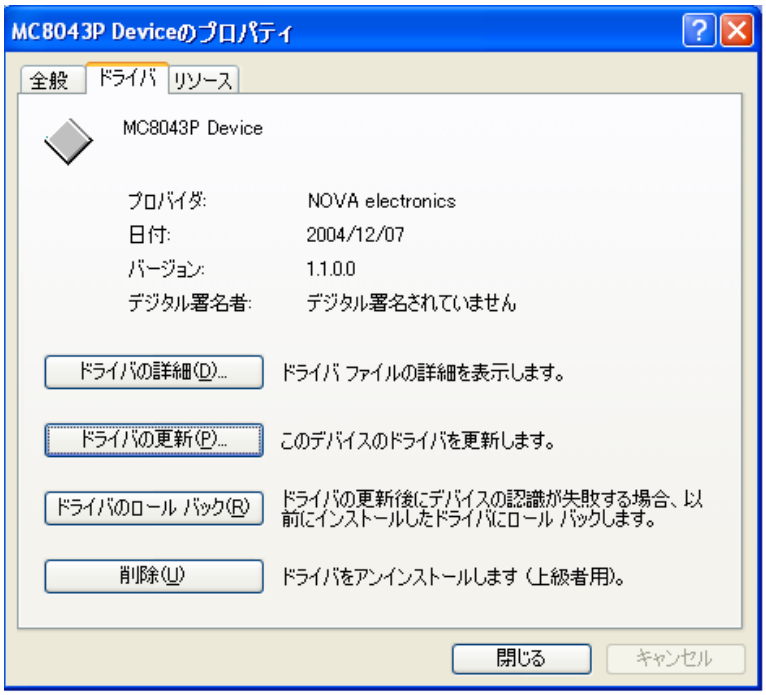
⑥下記画面で、「ディスク使用」をクリックし、参照ボタンから更新するドライバソフトのフォルダ（**¥Driver**）を選択後、下記画面の[次へ]をクリックします。



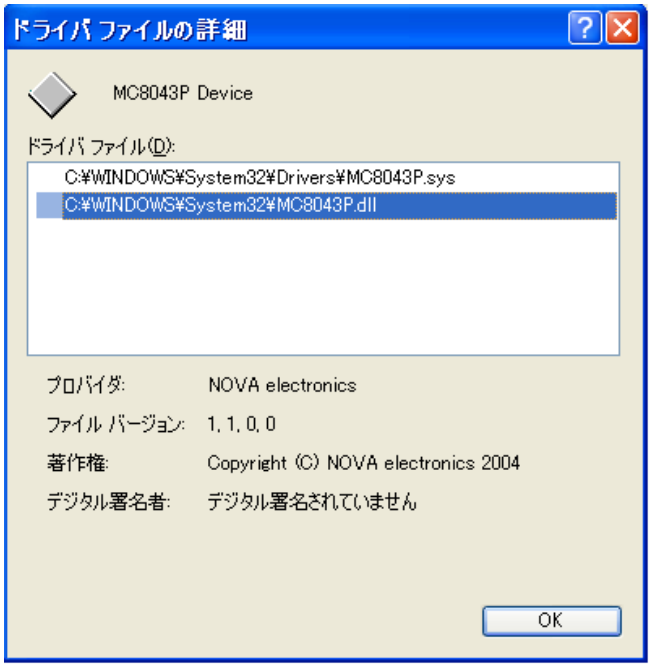
⑦正常に更新されると、下記画面が表示されます。



⑧下記画面（**MC8043P Device** のプロパティ画面）で、更新したドライバのバージョンを確認します。
¥Driver¥Version.txt ファイルの「１．ドライババージョン」のバージョンが下記画面に表示されますので正しいか確認して下さい。 次に「ドライバの詳細」ボタンをクリックします。



⑨下記画面で、更新したドライバのファイルバージョンを確認します。
¥Driver¥Version.txt ファイルの「２．ドライバファイルバージョン」のバージョンが下記画面に表示されます。
Version.txt ファイルに記載している２つのファイルのバージョンが正しいか確認して下さい。



⑩ **MC8043P** ボードが複数ある場合は、「デバイスマネージャ」画面の「ＮＯＶＡ」の下に表示される全ての「ＭＣ８０４３Ｐ Ｄｅｖｉｃｅ」について、①からの手順でドライバを更新して下さい。

⑪パソコンを再起動して下さい。これで更新作業は終了です。

8.6 外部機器との接続時の注意

本製品を外部機器と接続して動作させる場合には以下の点に注意してください。
□出力信号は出力信号同士や他の機器の出力信号と接続しないで下さい。故障の原因となります。
□出力信号を外部電源と短絡すると故障の原因となります。
□誤動作時の安全確保のため必ずリミットスイッチ入力を本ボードと外部機器間で接続してください。
□モータを駆動する前に必ず配線などに間違いがないか十分に確認してください。
モータと装置を切り離した状態でモータの回転、リミットスイッチの動作を必ず確認してからご使用ください。
□サージ電圧の入力は本ボードの故障の原因となる場合があります。
□ 入／出力信号の接続
外部電源や入力／出力信号の接続において、極性を逆にしたり、定格範囲を越えた電圧／電流を印加すると、回路素子を破壊したり、動作の信頼性を低下させる原因となります。十分配線を確認の上、接続してください。
□ Ｉ／Ｏケーブルの処置
付属のＩ／Ｏケーブルは１．２ｍの長さですが、A33～A50、およびB33～B50の信号はパソコン内部と同じ回路系の入出力信号線ですので、周囲からの電磁誘導ノイズを極力受けないよう十分配慮し、必要最小の長さでご使用ください。

9. プログラミング

この章では、アプリケーション開発のためのソフトウェア仕様とプログラミング方法について説明します。
アプリケーション開発は、Microsoft Visual C++ (以下VC++)、Microsoft Visual Basic(以下VB)、あるいMicrosoft Visual C# (以下C#)のいずれかを使用して行います。

9.1 ソフトウェアの仕様

9.1.1 動作環境

対応ＯＳ　Windows98 Windows2000 WindowsXP

対応言語　Microsoft Visual C++ 6.0
Microsoft Visual Basic 6.0

Microsoft Visual C++　.NET 2003 (Windows2000, WindowsXP)
Microsoft Visual Basic .NET 2003 (Windows2000, WindowsXP)
Microsoft Visual C#　.NET 2003 (Windows2000, WindowsXP)

9.1.2 プログラム構成

プログラム構成 種類	フォルダ	ﾌｧｲﾙ名・ﾌｫﾙﾀﾞ名	説　　明
デバイスドライバ	Driver	MC8043P.SYS	デバイスドライバ本体
		MC8043P.DLL	ダイナミックリンクライブラリ VC++,VB共通
		MC8043P.INF	インストールファイル
ライブラリ	Lib¥VB6	MC8043P_DLL.BAS	MC8043P.DLLを使用する為のDeclare宣言ﾌｧｲﾙ VB6.0専用
	Lib¥VB.NET2003	MC8043P_DLL.vb	MC8043P.DLLを使用する為のDeclare宣言ﾌｧｲﾙ VB.NET2003専用
	Lib¥VC6	MC8043P.LIB	MC8043P.DLLを使用するためのライブラリ VC++専用
		MC8043P_DLL.H	MC8043P.DLLを使用するためのヘッダ定義ファイル VC++専用
	Lib¥Csharp	Mc8043pWrap.dll	MC8043P.DLLを使用するためのクラスライブラリ .NET共通
VBサンプルプログラム (VB6.0)	Sample¥ VB6	Sample A	リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例
		Sample C	リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例 複数枚対応
		Sample E	全軸定量ドライブ、RR0,1,2,4,5読み出し例
		Sample G	補間関数を使用したＢＰ補間、連続補間例
		Sample H	自動原点出しのプログラム例
		Sample I	同期動作のプログラム例
		NormallyClose¥ Sample A	リミットセンサー論理 NormallyClose用 リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例
VBサンプルプログラム (VB.NET2003)	Sample¥ VB.NET2003	Sample A	リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例
		Sample C	リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例 複数枚対応
		Sample E	全軸定量ドライブ、RR0,1,2,4,5読み出し例
		Sample G	補間関数を使用したＢＰ補間、連続補間例
		Sample H	自動原点出しのプログラム例
		Sample I	同期動作のプログラム例
		NormallyClose¥ Sample A	リミットセンサー論理 NormallyClose用 リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例
VC++サンプルプログラム (VC6.0)	Sample¥ VC6	Sample A	リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例
		Sample B	割り込みを使用したプログラム例
		Sample C	リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例 複数枚対応
		Sample D	割り込みを使用したプログラム例　　複数枚対応
		Sample E	全軸定量ドライブ、RR0,1,2,4,5読み出し、割込みプログラム例
		Sample F	割込みを用いた連続補間のプログラム例
		Sample G	補間関数を使用したＢＰ補間、連続補間例
		Sample H	自動原点出しのプログラム例
		Sample I	同期動作のプログラム例
		NormallyClose¥ Sample A	リミットセンサー論理 NormallyClose用 リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例

プログラム構成 種類	フォルダ	ﾌｧｲﾙ名・ﾌｫﾙﾀﾞ名	説　　明
C#サンプルプログラム （C#.NET2003）	Sample¥ Csharp	Sample A	リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例
		Sample B	割り込みを使用したプログラム例
		Sample C	リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例 複数枚対応
		Sample D	割り込みを使用したプログラム例　　複数枚対応
		Sample E	全軸定量ドライブ、RR0,1,2,4,5読み出し、割込みプログラム例
		Sample F	割込みを用いた連続補間のプログラム例
		Sample G	補間関数を使用したＢＰ補間、連続補間例
		Sample H	自動原点出しのプログラム例
		Sample I	同期動作のプログラム例
MC8043P評価ツール	Tool	Tool8043.exe	画面からパラメータ、モード等を設定し、各コマンドを実行するアプリケーション
		ParameterSample	パラメータサンプルファイル： Tool8043.exeにてこのファイルをロードし、プロットを実行すると、各ファイル名称の動作をプロット画面上で確認できる

備考：VC++のMFC AppWizerdが自動的に作成するファイルに関しては説明を省略いたします。

9.1.3 ＡＰＩ（ MC8043Pドライバ関数 ）

MC8043P.SYS、MC8043P.DLL がアプリケーションに提供するＡＰＩ

9.1.3.1 関数一覧

下表は、API関数の一覧表です。
「VC」「VB」「VB.NET」「C#.NET」の欄は、各言語において各関数が使用できるかどうかを記載しています。
○は使用できます。×は使用できません。

（１）基本関数

関数名	説明	VC	VB	VB.NET	C#.NET	ページ
OpenMC8043P	MC8043Pの使用を開始する	○	○	○	○	51
CloseMC8043P	MC8043Pの使用を終了する	○	○	○	○	〃
CloseAllMC8043P	全てのMC8043Pの使用を終了する	○	○	○	○	〃
OutpMC8043P	出力ポートにデータを書く	○	○	○	○	52
InpMC8043P	入力ポートからデータを読む	○	○	○	○	〃
SetEventMC8043P	割込みを処理するユーザー関数を設定する	○	×	×	○	53
ResetEventMC8043P	割込みを処理するユーザー関数の設定を解除する	○	×	×	○	〃
ReadEventMC8043P	割込み発生直後の各軸ＲＲ３の値を取得する	○	×	×	○	54

（２）リセット、命令

関数名	説明	VC	VB	VB.NET	C#.NET	ページ
Nmc_Reset	MC8043Pをリセットする	○	○	○	○	55
Nmc_Command	指定軸の命令を実行する	○	○	○	○	〃
Nmc_Command_IP	補間命令を実行する	○	○	○	○	〃

（３）ライトレジスタ

関数名	説明	VC	VB	VB.NET	C#.NET	ページ
Nmc_WriteReg0	ＷＲ０(コマンドレジスタ)　　書き込み	○	○	○	○	56
Nmc_WriteReg1	ＷＲ１(モードレジスタ１)　　書き込み	○	○	○	○	〃
Nmc_WriteReg2	ＷＲ２(モードレジスタ２)　　書き込み	○	○	○	○	〃
Nmc_WriteReg3	ＷＲ３(モードレジスタ３)　　書き込み	○	○	○	○	57
Nmc_WriteReg4	ＷＲ４(アウトプットレジスタ)　書き込み	○	○	○	○	〃
Nmc_WriteReg5	ＷＲ５(補間モードレジスタ)　　書き込み	○	○	○	○	〃
Nmc_WriteReg6	ＷＲ６(ライトデータレジスタ１)書き込み	○	○	○	○	58
Nmc_WriteReg7	ＷＲ７(ライトデータレジスタ２)書き込み	○	○	○	○	〃

（４）リードレジスタ

関数名	説明	VC	VB	VB.NET	C#.NET	ページ
Nmc_ReadReg0	ＲＲ０(主ステータスレジスタ)　読み出し	○	○	○	○	58
Nmc_ReadReg1	ＲＲ１(ステータスレジスタ１)　読み出し	○	○	○	○	59
Nmc_ReadReg2	ＲＲ２(ステータスレジスタ２)　読み出し	○	○	○	○	〃
Nmc_ReadReg4	ＲＲ４(インプットレジスタ１)　読み出し	○	○	○	○	〃
Nmc_ReadReg5	ＲＲ５(インプットレジスタ２)　読み出し	○	○	○	○	60
Nmc_ReadReg6	ＲＲ６(リードデータレジスタ１)読み出し	○	○	○	○	〃
Nmc_ReadReg7	ＲＲ７(リードデータレジスタ２)読み出し	○	○	○	○	〃

（５）パラメータ設定

関数名	説明	VC	VB	VB.NET	C#.NET	ページ
Nmc_Range	レンジ設定	○	○	○	○	61
Nmc_Jerk	加速度増加率設定	○	○	○	○	〃
Nmc_Acc	加速度設定	○	○	○	○	〃
Nmc_Dec	減速度設定	○	○	○	○	62
Nmc_StartSpd	初速度設定	○	○	○	○	〃
Nmc_Speed	ドライブ速度設定	○	○	○	○	63
Nmc_Pulse	出力パルス数／補間終点設定(VC,C#用)	○	×	×	○	〃
Nmc_Pulse_VB	出力パルス数／補間終点設定(VB用)	×	○	○	×	64
Nmc_DecP	マニュアル減速点設定(VC,C#用)	○	×	×	○	〃
Nmc_DecP_VB	マニュアル減速点設定(VB用)	×	○	○	×	〃
Nmc_Center	円弧中心点設定	○	○	○	○	65
Nmc_Lp	論理位置カウンタ設定	○	○	○	○	〃
Nmc_Ep	実位置カウンタ設定	○	○	○	○	〃
Nmc_CompP	ＣＯＭＰ＋レジスタ設定	○	○	○	○	66
Nmc_CompM	ＣＯＭＰ－レジスタ設定	○	○	○	○	〃
Nmc_AccOfst	加速カウンタオフセット設定	○	○	○	○	〃
Nmc_DJerk	減速度増加率設定	○	○	○	○	67
Nmc_HomeSpd	原点検出速度設定	○	○	○	○	〃

（６）拡張・同期動作モード設定

関数名	説明	VC	VB	VB.NET	C#.NET	ページ
Nmc_ExpMode	拡張モード設定	○	○	○	○	67
Nmc_SyncMode	同期動作モード設定	○	○	○	○	68

（７）データ読み出し

関数名	説明	VC	VB	VB.NET	C#.NET	ページ
Nmc_ReadLp	論理位置カウンタ読み出し	○	○	○	○	68
Nmc_ReadEp	実位置カウンタ読み出し	○	○	○	○	〃
Nmc_ReadSpeed	現在ドライブ速度読み出し	○	○	○	○	69
Nmc_ReadAccDec	現在加／減速度読み出し	○	○	○	○	〃
Nmc_ReadSyncBuff	同期動作バッファレジスタ読み出し	○	○	○	○	〃

（８）状態取得

関数名	説明	VC	VB	VB.NET	C#.NET	ページ
Nmc_GetDriveStatus	ドライブ状態取得	○	○	○	○	70
Nmc_GetCNextStatus	連続補間次データ書込み可能状態取得	○	○	○	○	〃
Nmc_GetBpSc	ＢＰ補間スタックカウンタ取得	○	○	○	○	71

（９）書き込み・読み出し

関数名	説明	VC	VB	VB.NET	C#.NET	ページ
Nmc_WriteRegSetAxis	軸指定ライトレジスタ書き込み(WR1～3)	○	○	○	○	71
Nmc_ReadRegSetAxis	軸指定リードレジスタ読み出し(RR1～2)	○	○	○	○	〃
Nmc_WriteData	データ書き込み（パラメータ）	○	○	○	○	72
Nmc_WriteData2	データ書き込み（拡張モード、同期動作モード）	○	○	○	○	〃
Nmc_ReadData	データ読み出し	○	○	○	○	〃

（１０）補間実行

関数名	説明	VC	VB	VB.NET	C#.NET	ページ
Nmc_2BPExec	２軸ＢＰ補間実行	○	○	○	○	73
Nmc_3BPExec	３軸ＢＰ補間実行	○	○	○	○	75
Nmc_2BPExec_BG	２軸ＢＰ補間実行（バックグラウンドで実行）	○	○	○	○	77
Nmc_3BPExec_BG	３軸ＢＰ補間実行（バックグラウンドで実行）	○	○	○	○	80
Nmc_2CIPExec	２軸連続補間実行	○	○	○	○	83
Nmc_3CIPExec	３軸連続補間実行	○	○	○	○	85
Nmc_2CIPExec_BG	２軸連続補間実行（バックグラウンドで実行）	○	○	○	○	87
Nmc_3CIPExec_BG	３軸連続補間実行（バックグラウンドで実行）	○	○	○	○	90
Nmc_IPStop	補間実行を中断する	○	○	○	○	93

9.1.3.2 関数仕様

VC++の場合・・・・**VC** と[VC]の項目を参照して下さい。
VC++.NETの場合・・**VC** と[VC]の項目を参照して下さい。
VBの場合・・・・・**VB** と[VB]の項目を参照して下さい。
VB.NETの場合・・・**VB.NET** と[VB]の項目を参照して下さい。
C#.NETの場合・・・**C#.NET** と[C#]の項目を参照して下さい。
指定のない項目は、各言語共通の内容です。

関数名	機能　及び　内容
OpenMC8043P	MC8043Pの使用を開始する。 **VC**　BOOL　OpenMC8043P(int No); **VB**　Function OpenMC8043P(ByVal No As Long) As Long **VB.NET**　Function OpenMC8043P(ByVal No As Integer) As Integer **C#.NET**　bool　MC8043P.OpenMC8043P(int No); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） **戻り値** [VC]　オープンに成功するとTRUE、失敗するとFALSE [VB]　オープンに成功すると０以外、失敗すると０ [C#]　オープンに成功するとtrue、失敗するとfalse **使用例** [VC]　status = OpenMC8043P(0);　　// ボード番号０をオープン [VB]　status = OpenMC8043P(0) [C#]　status = MC8043P.OpenMC8043P(0);
CloseMC8043P	MC8043Pの使用を終了する。 **VC**　BOOL　CloseMC8043P(int No); **VB**　Function CloseMC8043P(ByVal No As Long) As Long **VB.NET**　Function CloseMC8043P(ByVal No As Integer) As Integer **C#.NET**　bool　MC8043P.CloseMC8043P(int No); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） **戻り値** [VC]　クローズに成功するとTRUE、失敗するとFALSE [VB]　クローズに成功すると０以外、失敗すると０ [C#]　クローズに成功するとtrue、失敗するとfalse **使用例** [VC]　status = CloseMC8043P(0);　　// ボード番号０をクローズ [VB]　status = CloseMC8043P(0) [C#]　status = MC8043P.CloseMC8043P(0);
CloseAllMC8043P	全てのMC8043Pの使用を終了する。 **VC**　BOOL　CloseAllMC8043P(void); **VB**　Function CloseAllMC8043P() As Long **VB.NET**　Function CloseAllMC8043P() As Integer **C#.NET**　bool　MC8043P.CloseAllMC8043P(); **入力パラメータ** なし **戻り値** [VC]　クローズに成功するとTRUE、失敗するとFALSE [VB]　クローズに成功すると０以外、失敗すると０ [C#]　クローズに成功するとtrue、失敗するとfalse **使用例** [VC]　status = CloseAllMC8043P();　　// 全てのボードをクローズ [VB]　status = CloseAllMC8043P() [C#]　status = MC8043P.CloseAllMC8043P();

関数名	機能　及び　内容
OutpMC8043P	出力ポートに２バイトデータを書き込む。 **VC**　　void OutpMC8043P(int No, long Adr, long Data); **VB**　　Sub　OutpMC8043P(ByVal No As Long, ByVal Adr As Long, ByVal Data As Long) **VB.NET**　Sub　OutpMC8043P(ByVal No As Integer, ByVal Adr As Integer, ByVal Data As Integer) **C#.NET**　void MC8043P.OutpMC8043P(int No, REG_MCX314 Adr, int Data); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～9)） 　Adr　　書き込むアドレス。詳細は補足説明(1)参照。 　Data　書き込むデータ **戻り値** 　なし **使用例** 　[VC]　OutpMC8043P(No, MCX314_WR0, 0x8000);　　// ボードのソフトリセット 　[VB]　Call OutpMC8043P(No, MCX314_WR0, &H8000) 　[C#]　MC8043P.OutpMC8043P(No, REG_MCX314.WR0, 0x8000);
InpMC8043P	入力ポートから２バイトデータを読み出す。 **VC**　　long　　InpMC8043P(int No, long Adr); **VB**　　Function InpMC8043P(ByVal No As Long, ByVal adr As Long) As Long **VB.NET**　Function InpMC8043P(ByVal No As Integer, ByVal adr As Integer) As Integer **C#.NET**　int　　MC8043P.InpMC8043P(int No, REG_MCX314 Adr); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～9)） 　Adr　　読み出すアドレス。詳細は補足説明(1)参照。 **戻り値** 　入力ポートから読み込んだデータ **使用例** 　[VC]　data = InpMC8043P(0, MCX314_RR0);　　// リードレジスタ RR0 の読み出し 　[VB]　data = InpMC8043P(0, MCX314_RR0) 　[C#]　data = MC8043P.InpMC8043P(0, REG_MCX314.RR0); **注意** 　[VC]　ＲＲ３レジスタのデータ読み出しに関しては、ReadEventMC8043P関数の説明を参考にして下さい。

関数名	機能　及び　内容

SetEventMC8043P	割込みを処理するユーザー関数を設定する。
	この関数を実行すると、割り込みが発生した時にユーザー関数が呼び出され、指定した引数が１つ渡されます(C#は、引数を指定できません)。このユーザー関数は１つのスレッドとして起動されます。
	
	VC　BOOL SetEventMC8043P(int No, LPTHREAD_START_ROUTINE UserThread, LPVOID lpParameter);
	VB　使用できません
	VB.NET　使用できません
	C#.NET　bool MC8043P.SetEventMC8043P(int No, UserThread Callback);
	
	入力パラメータ
	No　ボード番号（ボード上のロータリースイッチの値(0～9)）
	[VC] UserThread　ユーザー関数のアドレス。
	[VC] lpParameter　ユーザー関数スレッドに渡す１つの引数を指定する。
	スレッドで使用可能なポインタを設定して下さい。
	引数を使用しない場合は、NULL等で良い。
	[C#] Callback　ユーザーメソッド（引数なしメソッドのデリゲート型）。
	詳細は9.1.3.3使用方法を参照。
	戻り値
	[VC] 成功するとTRUE、失敗するとFALSE
	[C#] 成功するとtrue、失敗するとfalse
	
	使用例
	[VC]
	（ボード番号０の場合）　// 関数のアドレスと引数を設定
	status = SetEventMC8043P(0,(LPTHREAD_START_ROUTINE)MC8043P_EventProc0, NULL);
	Nmc_WriteReg1(0, AXIS_ALL, 0x8000);　// 停止時割込み発生（全軸）
	
	（ボード番号１の場合）　// 関数のアドレスと引数を設定
	status = SetEventMC8043P(1,(LPTHREAD_START_ROUTINE)MC8043P_EventProc1, lpParameter);
	Nmc_WriteReg1(1, AXIS_ALL, 0x8000);　// 停止時割込み発生（全軸）
	
	■割込みユーザー関数宣言例
	void MC8043P_EventProc0(void);
	void MC8043P_EventProc1(LPVOID lpParameter);
	
	[C#]
	// 割込みユーザーメソッド isr をデリゲート型変数に代入
	MC8043P.callback[0] = new MC8043P.UserThread(isr);
	// 割込みユーザーメソッドの設定
	MC8043P.SetEventMC8043P(no, MC8043P.callback[0]);

ResetEventMC8043P	割込みを処理するユーザー関数の設定を解除する。
	この関数を実行すると、割り込みが発生してもユーザー関数は呼び出されません。
	
	VC　BOOL ResetEventMC8043P(int No);
	VB　使用できません
	VB.NET　使用できません
	C#.NET　bool MC8043P.ResetEventMC8043P(int No);
	
	入力パラメータ
	No　ボード番号（ボード上のロータリースイッチの値(0～9)）
	
	戻り値
	[VC] 成功するとTRUE、失敗するとFALSE
	[C#] 成功するとtrue、失敗するとfalse
	
	使用例
	[VC] Nmc_WriteReg1(No, AXIS_ALL, 0x0000);　// 割込み禁止（全軸）
	status = ResetEventMC8043P(No);
	[C#] MC8043P.Nmc_WriteReg1(No, AXIS.ALL, 0x0000);　// 割込み禁止（全軸）
	status = MC8043P.ResetEventMC8043P(No);

関数名	機能　及び　内容
ReadEventMC8043P	割込み発生直後の各軸ＲＲ３の値を取得する。（ＲＲ３は読み出し後クリア） **VC**　BOOL ReadEventMC8043P(int No, long* Rr3X, long* Rr3Y, long* Rr3Z, long* Rr3U); **VB**　使用できません **VB.NET**　使用できません **C#.NET**　bool MC8043P.ReadEventMC8043P(int No, int* Rr3X, int* Rr3Y, int* Rr3Z, int* Rr3U); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） Rr3X　Ｘ軸のＲＲ３を格納する為のバッファへのポインタ Rr3Y　Ｙ軸のＲＲ３を格納する為のバッファへのポインタ Rr3Z　Ｚ軸のＲＲ３を格納する為のバッファへのポインタ Rr3U　Ｕ軸のＲＲ３を格納する為のバッファへのポインタ **戻り値** [VC]　成功するとTRUE、失敗するとFALSE [C#]　成功するとtrue、失敗するとfalse **使用例** [VC]　long Rr3X, Rr3Y, Rr3Z, Rr3U; ReadEventMC8043P(No, &Rr3X, &Rr3Y, &Rr3Z, &Rr3U); [C#]　int Rr3X,Rr3Y,Rr3Z,Rr3U ;　　// Ｘ軸、Ｙ軸、Ｚ軸、Ｕ軸 MC8043P.ReadEventMC8043P(No, &Rr3X, &Rr3Y, &Rr3Z, &Rr3U); **注意** MC8043Pが割込を発生した直後ドライバ内でＲＲ３を読み出してしまうのでＲＲ３はクリアされてしまいます。割込み発生直後のＲＲ３を確認する場合はこの関数を使用してください。 また、SetEventMC8043P、ResetEventMC8043P関数の実行とは関係なく、割込みが発生するとドライバは必ずRR3データを読み出し保存します。ドライバ内に保存されたRR3データは、ReadEventMC8043P関数を実行して読み出すとクリアされます。 ドライバ内のRR3データをクリアしたい時は、ReadEventMC8043P関数を実行して下さい。

関数名	機能　及び　内容
Nmc_Reset	MC8043Pをリセットする。 **VC**　　void Nmc_Reset(int No); **VB**　　Sub　Nmc_Reset(ByVal No As Long) **VB.NET**　Sub　Nmc_Reset(ByVal No As Integer) **C#.NET**　void MC8043P.Nmc_Reset(int No); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～9)） **戻り値** 　なし **使用例** 　[VC]　Nmc_Reset(0);　　// ボード番号０をリセットする 　[VB]　Call Nmc_Reset(0) 　[C#]　MC8043P.Nmc_Reset(0);
Nmc_Command	指定軸の命令を実行する。（WR0に指定軸の命令を書く） **VC**　　void Nmc_Command(int No, int Axis, int cmd); **VB**　　Sub　Nmc_Command(ByVal No As Long, ByVal Axis As Long, ByVal cmd As Long) **VB.NET**　Sub　Nmc_Command(ByVal No As Integer, ByVal Axis As Integer, ByVal cmd As Integer) **C#.NET**　void MC8043P.Nmc_Command(int No, AXIS axis, CMD cmd); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～9)） 　Axis　命令を実行する軸。AXIS_X, AXIS_Y 等を指定する。複数軸指定可能。詳細は補足説明(2)参照。 　cmd　　命令番号。定義ファイル(※１)内のコマンド定義の「ドライブ命令、その他の命令」の中から一つを指定する。＋方向定量ドライブの場合はCMD_F_DRV_Pを指定する。C#は補足説明(4)参照。 　　※１：[VC]MC8043P_DLL.H, [VB]MC8043P_DLL.bas, [VB.NET]MC8043P_DLL.vb **戻り値** 　なし **使用例** 　[VC]　Nmc_Command(No, AXIS_X, CMD_F_DRV_P);　　// Ｘ軸の＋方向定量ドライブを実行する 　[VB]　Call Nmc_Command(No, AXIS_X, CMD_F_DRV_P) 　[C#]　MC8043P.Nmc_Command(No, AXIS.X, CMD.CMD_F_DRV_P);
Nmc_Command_IP	補間命令を実行する。（WR0に補間命令を書く） **VC**　　void Nmc_Command_IP(int No, int cmd); **VB**　　Sub　Nmc_Command_IP(ByVal No As Long, ByVal cmd As Long) **VB.NET**　Sub　Nmc_Command_IP(ByVal No As Integer, ByVal cmd As Integer) **C#.NET**　void MC8043P.Nmc_Command_IP(int No, CMD cmd); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～9)） 　cmd　　命令番号。定義ファイル(※１)内のコマンド定義の「補間命令」の中から一つを指定する。 　　２軸直線補間ドライブの場合はCMD_IP_2STを指定する。C#は補足説明(4)参照。 　　※１：[VC]MC8043P_DLL.H, [VB]MC8043P_DLL.bas, [VB.NET]MC8043P_DLL.vb **戻り値** 　なし **使用例** 　[VC]　Nmc_WriteReg5(No, 0x0004);　　// 補間軸設定（主軸：Ｘ、第２軸：Ｙ） 　　Nmc_Command_IP(No, CMD_IP_2ST);　　// ２軸直線補間ドライブを実行する 　[VB]　Call Nmc_WriteReg5(No, &H0004)　　' 補間軸設定（主軸：Ｘ、第２軸：Ｙ） 　　Call Nmc_Command_IP(No, CMD_IP_2ST)　　' ２軸直線補間ドライブを実行する 　[C#]　MC8043P.Nmc_WriteReg5(No, 0x0004);　　// 補間軸設定（主軸：Ｘ、第２軸：Ｙ） 　　MC8043P.Nmc_Command_IP(No, CMD.CMD_IP_2ST);　　// ２軸直線補間ドライブを実行する

関数名	機能　及び　内容
Nmc_WriteReg0	ＷＲ０(コマンドレジスタ)にデータを書き込む。 **VC**　　void Nmc_WriteReg0(int No, long wdata); **VB**　　Sub Nmc_WriteReg0(ByVal No As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_WriteReg0(ByVal No As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_WriteReg0(int No, int wdata); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～9)） wdata　書き込むデータ **戻り値** なし **使用例** [VC]　Nmc_WriteReg0(No, 0x0120);　　// Ｘ軸の＋方向定量ドライブを実行する [VB]　Call Nmc_WriteReg0(No, &H120) [C#]　MC8043P.Nmc_WriteReg0(No, 0x0120);
Nmc_WriteReg1	ＷＲ１(モードレジスタ１)にデータを書き込む。 **VC**　　void Nmc_WriteReg1(int No, int Axis, long wdata); **VB**　　Sub Nmc_WriteReg1(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_WriteReg1(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_WriteReg1(int No, AXIS axis, int wdata); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを書き込む軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。詳細は補足説明(2)参照。 wdata　書き込むデータ **戻り値** なし **使用例** [VC]　Nmc_WriteReg1(No, AXIS_X, 0x0002);　　// ドライブ停止入力信号IN0有効(Ｘ軸) [VB]　Call Nmc_WriteReg1(No, AXIS_X, &H2) [C#]　MC8043P.Nmc_WriteReg1(No, AXIS.X, 0x0002);
Nmc_WriteReg2	ＷＲ２(モードレジスタ２)にデータを書き込む。 **VC**　　void Nmc_WriteReg2(int No, int Axis, long wdata); **VB**　　Sub Nmc_WriteReg2(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_WriteReg2(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_WriteReg2(int No, AXIS axis, int wdata); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを書き込む軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。詳細は補足説明(2)参照。 wdata　書き込むデータ **戻り値** なし **使用例** [VC]　Nmc_WriteReg2(No, AXIS_Y, 0x2000);　　// ALARM 有効(Ｙ軸) [VB]　Call Nmc_WriteReg2(No, AXIS_Y, &H2000) [C#]　MC8043P.Nmc_WriteReg2(No, AXIS.Y, 0x2000);

関数名	機能　及び　内容
Nmc_WriteReg3	ＷＲ３(モードレジスタ３)にデータを書き込む。 **VC**　　void Nmc_WriteReg3(int No, int Axis, long wdata); **VB**　　Sub Nmc_WriteReg3(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_WriteReg3(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_WriteReg3(int No, AXIS axis, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを書き込む軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。詳細は補足説明(2)参照。 wdata　書き込むデータ **戻り値** なし **使用例** [VC] Nmc_WriteReg3(No, AXIS_ALL, 0x0100);　　// 全軸汎用出力OUT4 Hiレベル出力 [VB] Call Nmc_WriteReg3(No, AXIS_ALL, &H0100) [C#] MC8043P.Nmc_WriteReg3(No, AXIS.ALL, 0x0100);
Nmc_WriteReg4	ＷＲ４(アウトプットレジスタ)にデータを書き込む。 **VC**　　void Nmc_WriteReg4(int No, long wdata); **VB**　　Sub Nmc_WriteReg4(ByVal No As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_WriteReg4(ByVal No As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_WriteReg4(int No, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） wdata　書き込むデータ **戻り値** なし **使用例** [VC] Nmc_WriteReg4(No, 0x0001);　　// Ｘ軸汎用出力OUT0 Hiレベル出力 [VB] Call Nmc_WriteReg4(No, &H0001) [C#] MC8043P.Nmc_WriteReg4(No, 0x0001);
Nmc_WriteReg5	ＷＲ５(補間モードレジスタ)にデータを書き込む。 **VC**　　void Nmc_WriteReg5(int No, long wdata); **VB**　　Sub Nmc_WriteReg5(ByVal No As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_WriteReg5(ByVal No As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_WriteReg5(int No, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） wdata　書き込むデータ **戻り値** なし **使用例** [VC] Nmc_WriteReg5(No, 0x0024);　　// 補間軸設定(主軸：Ｘ、第２軸：Ｙ、第３軸：Ｚ) [VB] Call Nmc_WriteReg5(No, &H0024) [C#] MC8043P.Nmc_WriteReg5(No, 0x0024);

関数名	機能　及び　内容
Nmc_WriteReg6	ＷＲ６(ライトデータレジスタ１)にデータを書き込む。 **VC**　　void Nmc_WriteReg6(int No, long wdata); **VB**　　Sub　Nmc_WriteReg6(ByVal No As Long, ByVal wdata As Long) **VB.NET**　Sub　Nmc_WriteReg6(ByVal No As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_WriteReg6(int No, int wdata); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～9)） 　wdata　書き込むデータ **戻り値** 　なし **使用例** 　[VC]　Nmc_WriteReg6(No, 0x1234);　　// ライトデータレジスタ１にデータ(1234)Hを書く 　[VB]　Call Nmc_WriteReg6(No, &H1234) 　[C#]　MC8043P.Nmc_WriteReg6(No, 0x1234);
Nmc_WriteReg7	ＷＲ７(ライトデータレジスタ２)にデータを書き込む。 **VC**　　void Nmc_WriteReg7(int No, long wdata); **VB**　　Sub　Nmc_WriteReg7(ByVal No As Long, ByVal wdata As Long) **VB.NET**　Sub　Nmc_WriteReg7(ByVal No As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_WriteReg7(int No, int wdata); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～9)） 　wdata　書き込むデータ **戻り値** 　なし **使用例** 　[VC]　Nmc_WriteReg7(No, 0x5678);　　// ライトデータレジスタ２にデータ(5678)Hを書く 　[VB]　Call Nmc_WriteReg7(No, &H5678) 　[C#]　MC8043P.Nmc_WriteReg7(No, 0x5678);
Nmc_ReadReg0	ＲＲ０(主ステータスレジスタ)のデータを読み出す。 **VC**　　long　　Nmc_ReadReg0(int No); **VB**　　Function Nmc_ReadReg0(ByVal No As Long) As Long **VB.NET**　Function Nmc_ReadReg0(ByVal No As Integer) As Integer **C#.NET**　int　　MC8043P.Nmc_ReadReg0(int No); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～9)） **戻り値** 　ＲＲ０(主ステータスレジスタ)のデータ **使用例** 　[VC]　Data = Nmc_ReadReg0(No);　　// ＲＲ０のデータを読む 　[VB]　Data = Nmc_ReadReg0(No) 　[C#]　Data = MC8043P.Nmc_ReadReg0(No);

関数名	機能　及び　内容
Nmc_ReadReg1	ＲＲ１(ステータスレジスタ１)のデータを読み出す。 **VC**　long　Nmc_ReadReg1(int No, int Axis); **VB**　Function Nmc_ReadReg1(ByVal No As Long, ByVal Axis As Long) As Long **VB.NET**　Function Nmc_ReadReg1(ByVal No As Integer, ByVal Axis As Integer) As Integer **C#.NET**　int　MC8043P.Nmc_ReadReg1(int No, AXIS axis); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを読み出す軸。詳細は補足説明(2)参照。 **戻り値** ＲＲ１(ステータスレジスタ１)のデータ **使用例** [VC]　Data = Nmc_ReadReg1(No, AXIS_X);　　// Ｘ軸のＲＲ１のデータを読む [VB]　Data = Nmc_ReadReg1(No, AXIS_X) [C#]　Data = MC8043P.Nmc_ReadReg1(No, AXIS.X);
Nmc_ReadReg2	ＲＲ２(ステータスレジスタ２)のデータを読み出す。 **VC**　long　Nmc_ReadReg2(int No, int Axis); **VB**　Function Nmc_ReadReg2(ByVal No As Long, ByVal Axis As Long) As Long **VB.NET**　Function Nmc_ReadReg2(ByVal No As Integer, ByVal Axis As Integer) As Integer **C#.NET**　int　MC8043P.Nmc_ReadReg2(int No, AXIS axis); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを読み出す軸。詳細は補足説明(2)参照。 **戻り値** ＲＲ２(ステータスレジスタ２)のデータ **使用例** [VC]　Data = Nmc_ReadReg2(No, AXIS_Y);　　// Ｙ軸のＲＲ２のデータを読む [VB]　Data = Nmc_ReadReg2(No, AXIS_Y) [C#]　Data = MC8043P.Nmc_ReadReg2(No, AXIS.Y);
Nmc_ReadReg4	ＲＲ４(インプットレジスタ１)のデータを読み出す。 **VC**　long　Nmc_ReadReg4(int No); **VB**　Function Nmc_ReadReg4(ByVal No As Long) As Long **VB.NET**　Function Nmc_ReadReg4(ByVal No As Integer) As Integer **C#.NET**　int　MC8043P.Nmc_ReadReg4(int No); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） **戻り値** ＲＲ４(インプットレジスタ１)のデータ **使用例** [VC]　Data = Nmc_ReadReg4(No);　　// ＲＲ４のデータを読む [VB]　Data = Nmc_ReadReg4(No) [C#]　Data = MC8043P.Nmc_ReadReg4(No);

関数名	機能　及び　内容
Nmc_ReadReg5	ＲＲ５(インプットレジスタ２)のデータを読み出す。 **VC**　　long　　Nmc_ReadReg5(int No); **VB**　　Function Nmc_ReadReg5(ByVal No As Long) As Long **VB.NET**　Function Nmc_ReadReg5(ByVal No As Integer) As Integer **C#.NET**　int　　MC8043P.Nmc_ReadReg5(int No); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～9)） **戻り値** 　ＲＲ５(インプットレジスタ２)のデータ **使用例** 　[VC]　Data = Nmc_ReadReg5(No);　　// ＲＲ５のデータを読む 　[VB]　Data = Nmc_ReadReg5(No) 　[C#]　Data = MC8043P.Nmc_ReadReg5(No);
Nmc_ReadReg6	ＲＲ６(リードデータレジスタ１)のデータを読み出す。 **VC**　　long　　Nmc_ReadReg6(int No); **VB**　　Function Nmc_ReadReg6(ByVal No As Long) As Long **VB.NET**　Function Nmc_ReadReg6(ByVal No As Integer) As Integer **C#.NET**　int　　MC8043P.Nmc_ReadReg6(int No); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～9)） **戻り値** 　ＲＲ６(リードデータレジスタ１)のデータ **使用例** 　[VC]　Data = Nmc_ReadReg6(No);　　// ＲＲ６のデータを読む 　[VB]　Data = Nmc_ReadReg6(No) 　[C#]　Data = MC8043P.Nmc_ReadReg6(No);
Nmc_ReadReg7	ＲＲ７(リードデータレジスタ２)のデータを読み出す。 **VC**　　long　　Nmc_ReadReg7(int No); **VB**　　Function Nmc_ReadReg7(ByVal No As Long) As Long **VB.NET**　Function Nmc_ReadReg7(ByVal No As Integer) As Integer **C#.NET**　int　　MC8043P.Nmc_ReadReg7(int No); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～9)） **戻り値** 　ＲＲ７(リードデータレジスタ２)のデータ **使用例** 　[VC]　Data = Nmc_ReadReg7(No);　　// ＲＲ７のデータを読む 　[VB]　Data = Nmc_ReadReg7(No) 　[C#]　Data = MC8043P.Nmc_ReadReg7(No);

関数名	機能　及び　内容
Nmc_Range	レンジを設定する。 **VC**　　void Nmc_Range(int No, int Axis, long wdata); **VB**　　Sub Nmc_Range(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_Range(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_Range(int No, AXIS axis, int wdata); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC]　Nmc_Range(No, AXIS_ALL, 800000);　　// レンジに 800000(倍率10)を設定する(全軸) [VB]　Call Nmc_Range(No, AXIS_ALL, 800000) [C#]　MC8043P.Nmc_Range(No, AXIS.ALL, 800000);
Nmc_Jerk	加速度増加率を設定する。 **VC**　　void Nmc_Jerk(int No, int Axis, long wdata); **VB**　　Sub Nmc_Jerk(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_Jerk(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_Jerk(int No, AXIS axis, int wdata); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC]　Nmc_Jerk(No, AXIS_X, 1000);　　// 加速度増加率に 1000 を設定する(Ｘ軸) [VB]　Call Nmc_Jerk(No, AXIS_X, 1000) [C#]　MC8043P.Nmc_Jerk(No, AXIS.X, 1000);
Nmc_Acc	加速度を設定する。 **VC**　　void Nmc_Acc(int No, int Axis, long wdata); **VB**　　Sub Nmc_Acc(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_Acc(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_Acc(int No, AXIS axis, int wdata); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC]　Nmc_Acc(No, AXIS_Y, 100);　　// 加速度に 100 を設定する(Ｙ軸) [VB]　Call Nmc_Acc(No, AXIS_Y, 100) [C#]　MC8043P.Nmc_Acc(No, AXIS.Y, 100);

関数名	機能　及び　内容
Nmc_Dec	減速度を設定する。 **VC**　　void Nmc_Dec(int No, int Axis, long wdata); **VB**　　Sub Nmc_Dec(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_Dec(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_Dec(int No, AXIS axis, int wdata); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC]　Nmc_Dec(No, AXIS_Z, 100);　　// 減速度に 100 を設定する(Z軸) [VB]　Call Nmc_Dec(No, AXIS_Z, 100) [C#]　MC8043P.Nmc_Dec(No, AXIS.Z, 100);
Nmc_StartSpd	初速度を設定する。 **VC**　　void Nmc_StartSpd(int No, int Axis, long wdata); **VB**　　Sub Nmc_StartSpd(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_StartSpd(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_StartSpd(int No, AXIS axis, int wdata); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC]　Nmc_StartSpd(No, AXIS_U, 100);　　// 初速度に 100 を設定する(U軸) [VB]　Call Nmc_StartSpd(No, AXIS_U, 100) [C#]　MC8043P.Nmc_StartSpd(No, AXIS.U, 100);

関数名	機能　及び　内容
Nmc_Speed	ドライブ速度を設定する。 **VC**　　void Nmc_Speed(int No, int Axis, long wdata); **VB**　　Sub Nmc_Speed(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_Speed(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_Speed(int No, AXIS axis, int wdata); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC]　Nmc_Speed(No, AXIS_X | AXIS_Y, 1000);　　// ドライブ速度に 1000 を設定する(X,Y軸) [VB]　Call Nmc_Speed(No, AXIS_X Or AXIS_Y, 1000) [C#]　MC8043P.Nmc_Speed(No, AXIS.X | AXIS.Y, 1000);

Nmc_Pulse	出力パルス数、あるいは補間終点を設定する。(VC,C#用)
	
	出力パルス数は、定量パルスドライブの総出力パルス数です。
	直線補間、円弧補間ドライブの時は、各軸の終点を設定します。
	終点座標は、現在位置に対する相対値を指定します。
	出力パルス数は符号無し32ビット、補間終点は符号有り32ビットの値をセットして下さい。
	
	VC　　void Nmc_Pulse(int No, int Axis, long wdata);
	VB　　使用できません
	VB.NET　使用できません
	C#.NET　補間終点(P)設定の場合
	void MC8043P.Nmc_Pulse(int No, AXIS axis, int wdata);
	出力パルス数設定の場合
	void MC8043P.Nmc_Pulse(int No, AXIS axis, uint wdata);
	
	入力パラメータ
	No　　ボード番号（ボード上のロータリースイッチの値(0～9)）
	Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。
	wdata　設定するデータ
	
	戻り値
	なし
	
	使用例
	[VC]　Nmc_Pulse(No, AXIS_X, 2000);　　// 出力パルス数に 2000 を設定する(X軸)
	
	Nmc_Pulse(No, AXIS_Y,　300);　　// 補間終点に　300 を設定する(Y軸)
	Nmc_Pulse(No, AXIS_Z, -400);　　// 補間終点に -400 を設定する(Z軸)
	
	[C#]　MC8043P.Nmc_Pulse(No, AXIS.X, 2000);
	MC8043P.Nmc_Pulse(No, AXIS.Y, 300);
	MC8043P.Nmc_Pulse(No, AXIS.Z, -400);

関数名	機能　及び　内容
Nmc_Pulse_VB	出力パルス数、あるいは補間終点を設定する。(VB専用) 出力パルス数は、定量パルスドライブの総出力パルス数です。 直線補間、円弧補間ドライブの時は、各軸の終点を設定します。 終点座標は、現在位置に対する相対値を指定します。 **VC**　使用できません **VB**　Sub Nmc_Pulse_VB(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Double) **VB.NET**　Sub Nmc_Pulse_VB(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Double) **C#.NET**　使用できません **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを設定する軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VB]　Call Nmc_Pulse_VB(No, AXIS_X, 2000)　　' 出力パルス数に 2000 を設定する(Ｘ軸) Call Nmc_Pulse_VB(No, AXIS_Y, 300)　　' 補間終点に 300 を設定する(Ｙ軸) Call Nmc_Pulse_VB(No, AXIS_Z, -400)　　' 補間終点に -400 を設定する(Ｚ軸)
Nmc_DecP	マニュアル減速点を設定する。(VC,C#用) **VC**　void Nmc_DecP(int No, int Axis, ULONG wdata); **VB**　使用できません **VB.NET**　使用できません **C#.NET**　void MC8043P.Nmc_DecP(int No, AXIS axis, uint wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC]　Nmc_DecP(No, AXIS_U, 30000);　　// マニュアル減速点に 30000 を設定する(Ｕ軸) [C#]　MC8043P.Nmc_DecP(No, AXIS.U, 30000);
Nmc_DecP_VB	マニュアル減速点を設定する。(VB専用) **VC**　使用できません **VB**　Sub Nmc_DecP_VB(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Double) **VB.NET**　Sub Nmc_DecP_VB(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Double) **C#.NET**　使用できません **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを設定する軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VB]　Call Nmc_DecP_VB(No, AXIS_X, 40000)　　' マニュアル減速点に 40000 を設定する(Ｘ軸)

関数名	機能　及び　内容
Nmc_Center	円弧中心点を設定する。 **VC**　　void Nmc_Center(int No, int Axis, long wdata); **VB**　　Sub Nmc_Center(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_Center(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_Center(int No, AXIS axis, int wdata); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC]　Nmc_Center(No, AXIS_Y, 1500);　　// 円弧中心点に 1500 を設定する(Y軸) [VB]　Call Nmc_Center(No, AXIS_Y, 1500) [C#]　MC8043P.Nmc_Center(No, AXIS.Y, 1500);
Nmc_Lp	論理位置カウンタを設定する。 **VC**　　void Nmc_Lp(int No, int Axis, long wdata); **VB**　　Sub Nmc_Lp(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_Lp(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_Lp(int No, AXIS axis, int wdata); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC]　Nmc_Lp(No, AXIS_ALL, 0);　　// 全軸の論理位置カウンタを0クリアする [VB]　Call Nmc_Lp(No, AXIS_ALL, 0) [C#]　MC8043P.Nmc_Lp(No, AXIS.ALL, 0);
Nmc_Ep	実位置カウンタを設定する。 **VC**　　void Nmc_Ep(int No, int Axis, long wdata); **VB**　　Sub Nmc_Ep(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_Ep(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_Ep(int No, AXIS axis, int wdata); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC]　Nmc_Ep(No, AXIS_ALL, 0);　　// 全軸の実位置カウンタを0クリアする [VB]　Call Nmc_Ep(No, AXIS_ALL, 0) [C#]　MC8043P.Nmc_Ep(No, AXIS.ALL, 0);

関数名	機能　及び　内容
Nmc_CompP	ＣＯＭＰ＋レジスタを設定する。 **VC**　　void Nmc_CompP(int No, int Axis, long wdata); **VB**　　Sub Nmc_CompP(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_CompP(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_CompP(int No, AXIS axis, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC]　Nmc_CompP(No, AXIS_X, 50000);　　// ＣＯＭＰ＋レジスタに 50000 を設定する(Ｘ軸) [VB]　Call Nmc_CompP(No, AXIS_X, 50000) [C#]　MC8043P.Nmc_CompP(No, AXIS.X, 50000);
Nmc_CompM	ＣＯＭＰ－レジスタを設定する。 **VC**　　void Nmc_CompM(int No, int Axis, long wdata); **VB**　　Sub Nmc_CompM(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_CompM(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_CompM(int No, AXIS axis, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを設定する軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC]　Nmc_CompM(No, AXIS_X, -50000);　　// ＣＯＭＰ－レジスタに -50000 を設定する(Ｘ軸) [VB]　Call Nmc_CompM(No, AXIS_X, -50000) [C#]　MC8043P.Nmc_CompM(No, AXIS.X, -50000);
Nmc_AccOfst	加速カウンタオフセットを設定する。 **VC**　　void Nmc_AccOfst(int No, int Axis, long wdata); **VB**　　Sub Nmc_AccOfst(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_AccOfst(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_AccOfst(int No, AXIS axis, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC]　Nmc_AccOfst(No, AXIS_Y, 20);　　// 加速カウンタオフセットに 20 を設定する(Ｙ軸) [VB]　Call Nmc_AccOfst(No, AXIS_Y, 20) [C#]　MC8043P.Nmc_AccOfst(No, AXIS.Y, 20);

関数名	機能　及び　内容
Nmc_DJerk	減速度増加率を設定する。 **VC**　　void Nmc_DJerk(int No, int Axis, long wdata); **VB**　　Sub Nmc_DJerk(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_DJerk(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_DJerk(int No, AXIS axis, int wdata); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC]　Nmc_DJerk(No, AXIS_Z, 1000);　　// 減速度増加率に 1000 を設定する(Z軸) [VB]　Call Nmc_DJerk(No, AXIS_Z, 1000) [C#]　MC8043P.Nmc_DJerk(No, AXIS.Z, 1000);
Nmc_HomeSpd	原点検出速度を設定する。 **VC**　　void Nmc_HomeSpd(int No, int Axis, long wdata); **VB**　　Sub Nmc_HomeSpd(ByVal No As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_HomeSpd(ByVal No As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_HomeSpd(int No, AXIS axis, int wdata); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC]　Nmc_HomeSpd(No, AXIS_U, 200);　　// 原点検出速度に 200 を設定する(U軸) [VB]　Call Nmc_HomeSpd(No, AXIS_U, 200) [C#]　MC8043P.Nmc_HomeSpd(No, AXIS.U, 200);
Nmc_ExpMode	拡張モードを設定する。 **VC**　　void Nmc_ExpMode(int No, int Axis, long EM6_data, long EM7_data); **VB**　　Sub Nmc_ExpMode(ByVal No As Long, ByVal Axis As Long, ByVal EM6_data As Long, ByVal EM7_data As Long) **VB.NET**　Sub Nmc_ExpMode(ByVal No As Integer, ByVal Axis As Integer, ByVal EM6_data As Integer, ByVal EM7_data As Integer) **C#.NET**　void MC8043P.Nmc_ExpMode(int No, AXIS axis, int EM6_data, int EM7_data); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 EM6_data　拡張モードレジスタ EM6 に設定するデータ EM7_data　拡張モードレジスタ EM7 に設定するデータ **戻り値** なし **使用例**　Ｘ軸拡張モードに、全フィルタ有効, 遅延512μs, 自動原点出しｽﾃｯﾌﾟ1,2,4の実行を設定する [VC]　Nmc_ExpMode(No, AXIS_X, 0x5F00, 0x0045); [VB]　Call Nmc_ExpMode(No, AXIS_X, &H5F00, &H0045) [C#]　MC8043P.Nmc_ExpMode(No, AXIS.X, 0x5F00, 0x0045);

関数名	機能　及び　内容
Nmc_SyncMode	同期動作モードを設定する。 **VC**　　void Nmc_SyncMode(int No, int Axis, long SM6_data, long SM7_data); **VB**　　Sub Nmc_SyncMode(ByVal No As Long, ByVal Axis As Long, ByVal SM6_data As Long, ByVal SM7_data As Long) **VB.NET**　Sub Nmc_SyncMode(ByVal No As Integer, ByVal Axis As Integer, ByVal SM6_data As Integer, ByVal SM7_data As Integer) **C#.NET**　void MC8043P.Nmc_SyncMode(int No, AXIS axis, int SM6_data, int SM7_data); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 SM6_data　同期動作モードレジスタ SM6 に設定するデータ SM7_data　同期動作モードレジスタ SM7 に設定するデータ **戻り値** なし **使用例**　「Ｘ軸ドライブ終了時にＹ軸＋方向定量ドライブを開始する」を設定する。 ① Ｘ軸同期動作モードに、起動要因 D-END でＹ軸起動を設定する ② Ｙ軸同期動作モードに、動作として＋方向定量ドライブ(FDRV+) を設定する [VC]　① Nmc_SyncMode(No, AXIS_X, 0x2020, 0); ② Nmc_SyncMode(No, AXIS_Y, 0, 0x0001); [VB]　① Call Nmc_SyncMode(No, AXIS_X, &H2020, 0) ② Call Nmc_SyncMode(No, AXIS_Y, 0, &H0001) [C#]　① MC8043P.Nmc_SyncMode(No, AXIS.X, 0x2020, 0); ② MC8043P.Nmc_SyncMode(No, AXIS.Y, 0, 0x0001);
Nmc_ReadLp	論理位置カウンタを読み出す。 **VC**　　long　　Nmc_ReadLp(int No, int Axis); **VB**　　Function Nmc_ReadLp(ByVal No As Long, ByVal Axis As Long) As Long **VB.NET**　Function Nmc_ReadLp(ByVal No As Integer, ByVal Axis As Integer) As Integer **C#.NET**　int　　MC8043P.Nmc_ReadLp(int No, AXIS axis); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを読み出す軸。詳細は補足説明(2)参照。 **戻り値** 現在の論理位置カウンタの値 **使用例** [VC]　Data = Nmc_ReadLp(No, AXIS_X);　　// Ｘ軸の論理位置カウンタを読み出す [VB]　Data = Nmc_ReadLp(No, AXIS_X) [C#]　Data = MC8043P.Nmc_ReadLp(No, AXIS.X);
Nmc_ReadEp	実位置カウンタを読み出す。 **VC**　　long　　Nmc_ReadEp(int No, int Axis); **VB**　　Function Nmc_ReadEp(ByVal No As Long, ByVal Axis As Long) As Long **VB.NET**　Function Nmc_ReadEp(ByVal No As Integer, ByVal Axis As Integer) As Integer **C#.NET**　int　　MC8043P.Nmc_ReadEp(int No, AXIS axis); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを読み出す軸。詳細は補足説明(2)参照。 **戻り値** 現在の実位置カウンタの値 **使用例** [VC]　Data = Nmc_ReadEp(No, AXIS_Y);　　// Ｙ軸の実位置カウンタを読み出す [VB]　Data = Nmc_ReadEp(No, AXIS_Y) [C#]　Data = MC8043P.Nmc_ReadEp(No, AXIS.Y)

関数名	機能　及び　内容
Nmc_ReadSpeed	現在ドライブ速度を読み出す。 **VC**　　long　　Nmc_ReadSpeed(int No, int Axis); **VB**　　Function Nmc_ReadSpeed(ByVal No As Long, ByVal Axis As Long) As Long **VB.NET**　Function Nmc_ReadSpeed(ByVal No As Integer, ByVal Axis As Integer) As Integer **C#.NET**　int　　MC8043P.Nmc_ReadSpeed(int No, AXIS axis); **入力パラメータ** 　No　ボード番号（ボード上のロータリースイッチの値(0～9)） 　Axis　データを読み出す軸。詳細は補足説明(2)参照。 **戻り値** 　現在ドライブ速度 **使用例** 　[VC]　Data = Nmc_ReadSpeed(No, AXIS_Z);　　// Ｚ軸の現在ドライブ速度を読み出す 　[VB]　Data = Nmc_ReadSpeed(No, AXIS_Z) 　[C#]　Data = MC8043P.Nmc_ReadSpeed(No, AXIS.Z);
Nmc_ReadAccDec	現在加／減速度を読み出す。 ドライブ中の現在加速度、または減速度の値を読み出す。 ドライブ停止時の読み出しデータは不定です。 **VC**　　long　　Nmc_ReadAccDec(int No, int Axis); **VB**　　Function Nmc_ReadAccDec(ByVal No As Long, ByVal Axis As Long) As Long **VB.NET**　Function Nmc_ReadAccDec(ByVal No As Integer, ByVal Axis As Integer) As Integer **C#.NET**　int　　MC8043P.Nmc_ReadAccDec(int No, AXIS axis); **入力パラメータ** 　No　ボード番号（ボード上のロータリースイッチの値(0～9)） 　Axis　データを読み出す軸。詳細は補足説明(2)参照。 **戻り値** 　現在加／減速度 **使用例** 　[VC]　Data = Nmc_ReadAccDec(No, AXIS_U);　　// Ｕ軸の現在加／減速度を読み出す 　[VB]　Data = Nmc_ReadAccDec(No, AXIS_U) 　[C#]　Data = MC8043P.Nmc_ReadAccDec(No, AXIS.U);
Nmc_ReadSyncBuff	同期動作バッファレジスタを読み出す。 **VC**　　long　　Nmc_ReadSyncBuff(int No, int Axis); **VB**　　Function Nmc_ReadSyncBuff(ByVal No As Long, ByVal Axis As Long) As Long **VB.NET**　Function Nmc_ReadSyncBuff(ByVal No As Integer, ByVal Axis As Integer) As Integer **C#.NET**　int　　MC8043P.Nmc_ReadSyncBuff(int No, AXIS axis); **入力パラメータ** 　No　ボード番号（ボード上のロータリースイッチの値(0～9)） 　Axis　データを読み出す軸。詳細は補足説明(2)参照。 **戻り値** 　同期動作バッファレジスタの値 **使用例** 　[VC]　Data = Nmc_ReadSyncBuff(No, AXIS_X);　// Ｘ軸の同期動作バッファレジスタを読み出す 　[VB]　Data = Nmc_ReadSyncBuff(No, AXIS_X) 　[C#]　Data = MC8043P.Nmc_ReadSyncBuff(No, AXIS.X);

関数名	機能　及び　内容

Nmc_GetDriveStatus	ドライブ状態を取得する。指定軸のドライブが終了したか調べる時に使用する。
	
	VC　　int　　Nmc_GetDriveStatus(int No, int Axis);
	VB　　Function Nmc_GetDriveStatus(ByVal No As Long, ByVal Axis As Long) As Long
	VB.NET　Function Nmc_GetDriveStatus(ByVal No As Integer, ByVal Axis As Integer) As Integer
	C#.NET　int　　MC8043P.Nmc_GetDriveStatus(int No, AXIS axis);
	
	入力パラメータ
	No　ボード番号（ボード上のロータリースイッチの値(0～9)）
	Axis　ドライブ状態を取得する軸。複数軸指定可能。詳細は補足説明(2)参照。
	
	戻り値
	指定した全ての軸のドライブが終了している場合は０を返す。
	指定した軸のうち１つ以上がドライブ中の場合は、０以外を返す。
	
	使用例
	[VC]　if(Nmc_GetDriveStatus(No, AXIS_X) == 0)　　// Ｘ軸がドライブ終了の場合
	AfxMessageBox("Ｘ軸ドライブ終了");
	else
	AfxMessageBox("Ｘ軸ドライブ中");
	[VB]　If Nmc_GetDriveStatus(No, AXIS_X) = 0 Then　　' Ｘ軸がドライブ終了の場合
	Call MsgBox("Ｘ軸ドライブ終了")
	Else
	Call MsgBox("Ｘ軸ドライブ中")
	End If
	[C#]　if(MC8043P.Nmc_GetDriveStatus(No, AXIS.X) == 0)　// Ｘ軸がドライブ終了の場合
	MessageBox.Show("Ｘ軸ドライブ終了");
	else
	MessageBox.Show("Ｘ軸ドライブ中");

Nmc_GetCNextStatus	連続補間次データ書込み可能状態を取得する。
	連続補間実行中に次データ書き込み可能になったか調べる時に使用する。
	
	VC　　int　　Nmc_GetCNextStatus(int No);
	VB　　Function Nmc_GetCNextStatus(ByVal No As Long) As Long
	VB.NET　Function Nmc_GetCNextStatus(ByVal No As Integer) As Integer
	C#.NET　int　　MC8043P.Nmc_GetCNextStatus(int No);
	
	入力パラメータ
	No　ボード番号（ボード上のロータリースイッチの値(0～9)）
	
	戻り値
	連続補間次データ書込み可能の場合は、０以外を返す。
	連続補間次データ書込み可能ではない場合は、０を返す。
	
	使用例
	[VC]　if(Nmc_GetCNextStatus(No) != 0)　　// 次データ書込み可能の場合
	AfxMessageBox("連続補間次データ書込み可能である");
	else
	AfxMessageBox("連続補間次データ書込み可能ではない");
	
	[VB]　If Nmc_GetCNextStatus(No) <> 0 Then　　' 次データ書込み可能の場合
	Call MsgBox("連続補間次データ書込み可能である")
	Else
	Call MsgBox("連続補間次データ書込み可能ではない")
	End If
	
	[C#]　if(MC8043P.Nmc_GetCNextStatus(No) != 0)　　// 次データ書込み可能の場合
	MessageBox.Show("連続補間次データ書込み可能である");
	else
	MessageBox.Show("連続補間次データ書込み可能ではない");

関数名	機能　及び　内容
Nmc_GetBpSc	ＢＰ補間スタックカウンタの値を取得する。 **VC**　　int　　Nmc_GetBpSc(int No); **VB**　　Function Nmc_GetBpSc(ByVal No As Long) As Long **VB.NET**　Function Nmc_GetBpSc(ByVal No As Integer) As Integer **C#.NET**　int　　MC8043P.Nmc_GetBpSc(int No); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） **戻り値** 現在のビットパターン補間スタックカウンタの値 **使用例** [VC]　Data = Nmc_GetBpSc(No);　　// ＢＰ補間スタックカウンタの値を取得 [VB]　Data = Nmc_GetBpSc(No) [C#]　Data = MC8043P.Nmc_GetBpSc(No);
Nmc_WriteRegSetAxis	指定軸の指定したライトレジスタ(ＷＲ１～ＷＲ３のいずれか)にデータを書き込む。 **VC**　　void Nmc_WriteRegSetAxis(int No, int Axis, int RegNumber, long wdata); **VB**　　Sub　Nmc_WriteRegSetAxis(ByVal No As Long, ByVal Axis As Long, ByVal RegNumber As Long, ByVal wdata As Long) **VB.NET**　Sub　Nmc_WriteRegSetAxis(ByVal No As Integer, ByVal Axis As Integer, ByVal RegNumber As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_WriteRegSetAxis(int No, AXIS axis, int RegNumber, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを書き込む軸。複数軸指定可能。詳細は補足説明(2)参照。 RegNumber　データを書き込むライトレジスタの番号。詳細は補足説明(1)参照。 wdata　書き込むデータ **戻り値** なし **使用例**　全軸のＷＲ２にALARM有効(2000)Hを書き込む [VC]　Nmc_WriteRegSetAxis(No, AXIS_ALL, MCX314_WR2, 0x2000); [VB]　Call Nmc_WriteRegSetAxis(No, AXIS_ALL, MCX314_WR2, &H2000) [C#]　MC8043P.Nmc_WriteRegSetAxis(No, AXIS.ALL, REG_MCX314.WR2, 0x2000);
Nmc_ReadRegSetAxis	指定軸の指定したリードレジスタ(ＲＲ１、ＲＲ２のどちらか)のデータを読み出す。 **VC**　　long　　Nmc_ReadRegSetAxis(int No, int Axis, int RegNumber); **VB**　　Function Nmc_ReadRegSetAxis(ByVal No As Long, ByVal Axis As Long, ByVal RegNumber As Long) As Long **VB.NET**　Function Nmc_ReadRegSetAxis(ByVal No As Integer, ByVal Axis As Integer, ByVal RegNumber As Integer) As Integer **C#.NET**　int　　MC8043P.Nmc_ReadRegSetAxis(int No, AXIS axis, int RegNumber); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを読み出す軸。詳細は補足説明(2)参照。 RegNumber　データを読み出すリードレジスタの番号。 詳細は補足説明(1)参照。 **戻り値** 指定軸の指定したリードレジスタのデータ **使用例**　Ｘ軸のＲＲ１のデータを読み出す [VC]　Data = Nmc_ReadRegSetAxis(No, AXIS_X, MCX314_RR1); [VB]　Data = Nmc_ReadRegSetAxis(No, AXIS_X, MCX314_RR1) [C#]　Data = MC8043P.Nmc_ReadRegSetAxis(No, AXIS.X, REG_MCX314.RR1);

関数名	機能　及び　内容
Nmc_WriteData	指定軸に指定したパラメータのデータを書き込む。(データ書き込み命令を実行する) **VC**　　void Nmc_WriteData(int No, int Axis, int cmd, long wdata); **VB**　　Sub Nmc_WriteData(ByVal No As Long, ByVal Axis As Long, ByVal cmd As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_WriteData(ByVal No As Integer, ByVal Axis As Integer, ByVal cmd As Integer, ByVal wdata As Integer) **C#.NET**　void MC8043P.Nmc_WriteData(int No, AXIS axis, CMD cmd, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを書き込む軸。複数軸指定可能。詳細は補足説明(2)参照。 cmd　データ書き込み命令コード。詳細は補足説明(1)参照。 wdata　書き込むデータ **戻り値** なし **使用例**　全軸のドライブ速度に1000を設定する。ドライブ速度の命令コードは(05)H [VC] Nmc_WriteData(No, AXIS_ALL, 0x05, 1000); [VB] Call Nmc_WriteData(No, AXIS_ALL, &H05, 1000) [C#] MC8043P. Nmc_WriteData(No, AXIS.ALL, CMD.CMD_Speed, 1000);
Nmc_WriteData2	指定軸に拡張モード、あるいは同期動作モードのデータを書き込む。(データ書き込み命令を実行する) **VC**　　void Nmc_WriteData2(int No, int Axis, int cmd, long WR6_data, long WR7_data); **VB**　　Sub Nmc_WriteData2(ByVal No As Long, ByVal Axis As Long, ByVal cmd As Long, ByVal WR6_data As Long, ByVal WR7_data As Long) **VB.NET**　Sub Nmc_WriteData2(ByVal No As Integer, ByVal Axis As Integer, ByVal cmd As Integer, ByVal WR6_data As Integer, ByVal WR7_data As Integer) **C#.NET**　void MC8043P.Nmc_WriteData2(int No, AXIS axis, CMD cmd, int WR6_data, int WR7_data); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを書き込む軸。複数軸指定可能。詳細は補足説明(2)参照。 cmd　拡張モードまたは同期動作モードのデータ書き込み命令コード。詳細は補足説明(1)参照。 WR6_data　拡張モードはEM6に，同期動作モードはSM6に書き込むデータ WR7_data　拡張モードはEM7に，同期動作モードはSM7に書き込むデータ **戻り値** なし **使用例**　Ｘ軸の拡張モードに、EM6データ(5F00)H，EM7データ(45)Hを書き込む。 [VC] Nmc_WriteData2(No, AXIS_X, 0x60, 0x5F00, 0x0045); [VB] Call Nmc_WriteData2(No, AXIS_X, &H60, &H5F00, &H45) [C#] MC8043P. Nmc_WriteData2(No, AXIS.X, CMD.CMD_ExpMode, 0x5F00, 0x0045);
Nmc_ReadData	データ読み出し命令を実行し、データを読み出す。 **VC**　　long　　Nmc_ReadData(int No, int Axis, int cmd); **VB**　　Function Nmc_ReadData(ByVal No As Long, ByVal Axis As Long, ByVal cmd As Long) As Long **VB.NET**　Function Nmc_ReadData(ByVal No As Integer, ByVal Axis As Integer, ByVal cmd As Integer) As Integer **C#.NET**　int　　MC8043P.Nmc_ReadData(int No, AXIS axis, CMD cmd); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） Axis　データを読み出す軸。詳細は補足説明(2)参照。 cmd　データ読み出し命令コード。詳細は補足説明(1)参照。 **戻り値** 読み出したデータ **使用例** [VC] Data = Nmc_ReadData(No,AXIS_X, 0x10);　// Ｘ軸の論理位置カウンタを読み出す。 [VB] Data = Nmc_ReadData(No,AXIS_X, &H10) [C#] Data = MC8043P. Nmc_ReadData(No, AXIS.X, CMD.CMD_ReadLp);

関数名	機能　及び　内容

Nmc_2BPExec	指定した補間データで２軸ビットパターン補間を実行する。
	この関数は、補間処理が終了した後に制御を返します。補間が終了するまで制御が戻らないのでアプリケーションでスレッドを作成し、そのスレッドからコールする事を推奨します。
	
	VC　DWORD　Nmc_2BPExec(int No, DATA_2BP* pData2Bp, int DataCnt, int lpAxis,
	BOOL ContinueFlg = FALSE);
	
	VB　Function Nmc_2BPExec(ByVal No As Long, ByRef pData2Bp As DATA_2BP,
	ByVal DataCnt As Long, ByVal lpAxis As Long,
	ByVal ContinueFlg As Long) As Long
	
	VB.NET　Function Nmc_2BPExec(ByVal No As Integer, ByRef pData2Bp As DATA_2BP,
	ByVal DataCnt As Integer, ByVal lpAxis As Integer,
	ByVal ContinueFlg As Integer) As Integer
	
	C#.NET　Nmc_Status MC8043P.Nmc_2BPExec(int No, DATA_2BP[] pData2Bp, int DataCnt, int lpAxis,
	bool ContinueFlg);
	
	入力パラメータ
	No　ボード番号（ボード上のロータリースイッチの値(0～9)）
	pData2Bp　２軸ＢＰ補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_2BPのｱﾄﾞﾚｽ)。
	DATA_2BPに実行する補間データをセットし、アドレスを指定する。
	DATA_2BPについては補足説明(3)参照。
	DataCnt　２軸ＢＰ補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。
	lpAxis　補間を実行する軸。WR5のD0～D5(軸指定)の設定値と同じ値を指定する。補足説明(4)参照。
	ContinueFlg BP補間が途中で停止した（速度が速く次データのスタックが間に合わなかった）場合に続けるかどうかを設定する。
	[VC]　TRUE ：続ける、FALSE ：続けない。省略可能。省略時FALSE。
	[VB]　True ：続ける、False ：続けない
	[C#]　true ：続ける、false ：続けない
	戻り値
	補間処理が正常終了した場合は BP_END が返ります。
	補間処理でエラーが発生した場合は、下記エラーコードが返ります。
	[C#]の場合は補足説明(1)参照。
	■正常終了
	BP_END　BP補間処理正常終了
	
	■エラーコード
	BP_CNT_ERR　指定したデータ数が範囲外です
	BP_ALREADY_EXEC　既にＢＰ補間、あるいは連続補間が実行中です
	BP_STOP　BP補間が途中で停止した(速度が速く次データのスタックが間に合わなかった)
	BP_USER_STOP　BP補間実行中にユーザーが中断した
	BP_DRIVE_ERR　BP補間実行中にMC8043Pでエラーが発生した（RR0にエラー情報がセットされた）
	
	使用例
	[VC] // ２軸ＢＰ補間データ　BP1P,　BP1M,　BP2P,　BP2M
	DATA_2BP Data2Bp[2] = {{0x0000, 0x2BFF, 0xFFD4, 0x0000},
	{0xF6FE, 0x0000, 0x000F, 0x3FC0}};
	
	Nmc_WriteReg5(No, 0x04);　// 補間軸設定。主軸：Ｘ、第２軸：Ｙ
	Ret = Nmc_2BPExec(No, Data2Bp, 2, 0x04);　// ２軸ＢＰ補間実行。ﾃﾞｰﾀ数2, X,Y軸
	if(Ret == BP_END)　AfxMessageBox("正常終了");　// 戻り値正常
	
	Dim Data2Bp(1) As DATA_2BP　'２軸ＢＰ補間データ
	'２軸ＢＰ補間データ設定
	Data2Bp(0).Bp1p = &H0:　Data2Bp(0).Bp1m = &H2BFF
	Data2Bp(0).Bp2p = &HFFD4: Data2Bp(0).Bp2m = &H0
	Data2Bp(1).Bp1p = &HF6FE: Data2Bp(1).Bp1m = &H0
	Data2Bp(1).Bp2p = &HF:　Data2Bp(1).Bp2m = &H3FC0
	
	Call Nmc_WriteReg5(No, &H4)　'補間軸設定。主軸：Ｘ、第２軸：Ｙ
	Ret = Nmc_2BPExec(No, Data2Bp(0), 2, &H4, False)　'２軸ＢＰ補間実行。ﾃﾞｰﾀ数2, X,Y軸
	If Ret = BP_END Then　'戻り値正常
	Call MsgBox("正常終了")
	End If

```
[C#]
    DATA_2BP [] Data2Bp = new DATA_2BP[4];  // ２軸ＢＰ補間データ
    // 補間データ設定
    Data2Bp[0].Bp1p = 0xFF30;       // 1111 1111 0011 0000  BP1＋方向       10パルス
    Data2Bp[0].Bp1m = 0;            // 0000 0000 0000 0000  BP1－方向        0パルス
    Data2Bp[0].Bp2p = 0;            // 0000 0000 0000 0000  BP2＋方向        0パルス
    Data2Bp[0].Bp2m = 0x84FF;       // 1000 0100 1111 1111  BP2－方向       10パルス

    IpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y << 2;      // 補間軸

    MC8043P.Nmc_WriteReg5(No, IpAxis);              // 補間軸設定
    MC8043P.Nmc_StartSpd(No, AXIS.ALL, 1);          // 初速度設定
    MC8043P.Nmc_Speed(gBoardNo, AXIS.ALL, 1);       // ドライブ速度設定
    // ２軸ＢＰ補間実行
    Ret = MC8043P.Nmc_2BPExec(No, Data2Bp, DataCnt, IpAxis, ContinueFlg);
    if(Ret == Nmc_Status.BP_END)                    // 戻り値正常
```

関数名	機能　及び　内容
Nmc_3BPExec	指定した補間データで３軸ビットパターン補間を実行する。 この関数は、補間処理が終了した後に制御を返します。補間が終了するまで制御が戻らないのでアプリケーションでスレッドを作成し、そのスレッドからコールする事を推奨します。 **VC**　DWORD　Nmc_3BPExec(int No, DATA_3BP* pData3Bp, int DataCnt, int lpAxis, BOOL ContinueFlg = FALSE); **VB**　Function Nmc_3BPExec(ByVal No As Long, ByRef pData3Bp As DATA_3BP, ByVal DataCnt As Long, ByVal lpAxis As Long, ByVal ContinueFlg As Long) As Long **VB.NET**　Function Nmc_3BPExec(ByVal No As Integer, ByRef pData3Bp As DATA_3BP, ByVal DataCnt As Integer, ByVal lpAxis As Integer, ByVal ContinueFlg As Integer) As Integer **C#.NET**　Nmc_Status MC8043P.Nmc_3BPExec(int No, DATA_3BP[] pData3Bp, int DataCnt, int lpAxis, bool ContinueFlg); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） pData3Bp　３軸ＢＰ補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_3BPのｱﾄﾞﾚｽ)。 DATA_3BPに実行する補間データをセットし、アドレスを指定する。 DATA_3BPについては補足説明(3)参照。 DataCnt　３軸ＢＰ補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 lpAxis　補間を実行する軸。WR5のD0～D5(軸指定)の設定値と同じ値を指定する。補足説明(4)参照。 ContinueFlg BP補間が途中で停止した（速度が速く次データのスタックが間に合わなかった）場合に続けるかどうかを設定する。 [VC]　TRUE ：続ける、FALSE ：続けない。省略可能。省略時FALSE。 [VB]　True ：続ける、False ：続けない [C#]　true ：続ける、false ：続けない **戻り値** 補間処理が正常終了した場合は BP_END が返ります。 補間処理でエラーが発生した場合は、下記エラーコードが返ります。 [C#]の場合は補足説明(1)参照。 ■正常終了 BP_END　BP補間処理正常終了 ■エラーコード BP_CNT_ERR　指定したデータ数が範囲外です BP_ALREADY_EXEC　既にＢＰ補間、あるいは連続補間が実行中です BP_STOP　BP補間が途中で停止した(速度が速く次データのスタックが間に合わなかった) BP_USER_STOP　BP補間実行中にユーザーが中断した BP_DRIVE_ERR　BP補間実行中にMC8043Pでエラーが発生した（RR0にエラー情報がセットされた） **使用例** [VC] // ３軸ＢＰ補間データ　BP1P,　BP1M,　BP2P,　BP2M,　BP3P,　BP3M DATA_3BP Data3Bp[2] = {{0xFF30,　0,　0,　0x84FF,　0,　0xAC35}, {0xAC35,　0,　0xC000,　0x36E7,　0xC000,　0x3F3F}}; Nmc_WriteReg5(No, 0x24);　// 補間軸設定。主軸：X, 第2軸：Y, 第3軸：Z Ret = Nmc_3BPExec(No, Data3Bp, 2, 0x24);　// ３軸ＢＰ補間実行。ﾃﾞｰﾀ数2, X,Y,Z軸 if(Ret == BP_END)　AfxMessageBox("正常終了"); // 戻り値正常 [VB] Dim Data3Bp(1) As DATA_3BP　'３軸ＢＰ補間データ '３軸ＢＰ補間データ設定 Data3Bp(0).Bp1p = &HFF30: Data3Bp(0).Bp1m = &H0 Data3Bp(0).Bp2p = &H0:　Data3Bp(0).Bp2m = &H84FF Data3Bp(0).Bp3p = &H0:　Data3Bp(0).Bp3m = &HAC35 Data3Bp(1).Bp1p = &HAC35: Data3Bp(1).Bp1m = &H0 Data3Bp(1).Bp2p = &HC000: Data3Bp(1).Bp2m = &H36E7 Data3Bp(1).Bp3p = &HC000: Data3Bp(1).Bp3m = &H3F3F

```
    Call Nmc_WriteReg5(No, &H24)                        '補間軸設定。主軸：X, 第2軸：Y, 第3軸：Z
    Ret = Nmc_3BPExec(No, Data3Bp(0), 2, &H24, False)   '３軸ＢＰ補間実行。ﾃﾞｰﾀ数2, X,Y,Z軸
    If Ret = BP_END Then                                '戻り値正常
        Call MsgBox("正常終了")
    End If

[C#]
   DATA_3BP [] Data3Bp = new DATA_3BP[4];                    // ３軸ＢＰ補間データ
   // 補間データ設定
   Data3Bp[0].Bp1p = 0xFF30;        // 1111 1111 0011 0000  BP1＋方向       10パルス
   Data3Bp[0].Bp1m = 0;             // 0000 0000 0000 0000  BP1－方向        0パルス
   Data3Bp[0].Bp2p = 0;             // 0000 0000 0000 0000  BP2＋方向        0パルス
   Data3Bp[0].Bp2m = 0x84FF;        // 1000 0100 1111 1111  BP2－方向       10パルス
   Data3Bp[0].Bp3p = 0;             // 0000 0000 0000 0000  BP3＋方向        0パルス
   Data3Bp[0].Bp3m = 0xAC35;        // 1010 1100 0011 0101  BP3－方向        8パルス

   IpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y << 2 | IP_AXIS.IP_Z << 4;      // 補間軸

   MC8043P.Nmc_WriteReg5(No, IpAxis);                // 補間軸設定
   MC8043P.Nmc_StartSpd(No, AXIS.ALL, 1);            // 初速度設定
   MC8043P.Nmc_Speed(No, AXIS.ALL, 1);               // ドライブ速度設定
   // ３軸ＢＰ補間実行
   Ret = MC8043P.Nmc_3BPExec(No, Data3Bp, DataCnt, IpAxis, ContinueFlg);
   if(Ret == Nmc_Status.BP_END)                     // 戻り値正常
```

関数名	機能　及び　内容
Nmc_2BPExec_BG	指定した補間データで２軸ビットパターン補間をバックグラウンドで実行する。 この関数は、補間処理を開始した直後に制御を返し、バックグラウンドで補間を実行します。 指定したウィンドウに対して、補間終了時にWM_BP_ENDメッセージを送信し、終了ステータスを渡します。 **VC**　DWORD　Nmc_2BPExec_BG(HWND User_hWnd, int No, DATA_2BP* pData2Bp, int DataCnt, int IpAxis, BOOL ContinueFlg = FALSE); **VB**　Function Nmc_2BPExec_BG(ByVal User_hWnd As Long, ByVal No As Long, ByRef pData2Bp As DATA_2BP, ByVal DataCnt As Long, ByVal IpAxis As Long, ByVal ContinueFlg As Long) As Long **VB.NET**　Function Nmc_2BPExec_BG(ByVal User_hWnd As Integer, ByVal No As Integer, ByRef pData2Bp As DATA_2BP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal ContinueFlg As Integer) As Integer **C#.NET**　Nmc_Status MC8043P.Nmc_2BPExec_BG(System.IntPtr User_hWnd, int No, DATA_2BP[] pData2Bp, int DataCnt, int IpAxis, bool ContinueFlg); **入力パラメータ** User_hWnd　ユーザーアプリケーションのウィンドウハンドル No　ボード番号（ボード上のロータリースイッチの値(0～9)） pData2Bp　２軸ＢＰ補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_2BPのｱﾄﾞﾚｽ)。 DATA_2BPに実行する補間データをセットし、アドレスを指定する。 DATA_2BPについては補足説明(3)参照。 DataCnt　２軸ＢＰ補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 IpAxis　補間を実行する軸。WR5のD0～D5(軸指定)の設定値と同じ値を指定する。補足説明(4)参照。 ContinueFlg BP補間が途中で停止した（速度が速く次データのスタックが間に合わなかった）場合に 続けるかどうかを設定する。 [VC]　TRUE　：続ける、FALSE　：続けない。省略可能。省略時FALSE。 [VB]　True　：続ける、False　：続けない [C#]　true　：続ける、false　：続けない **戻り値** バックグラウンドで補間処理が正常に開始した場合は BP_START が返ります。 補間処理が開始する前にエラーが発生した場合は、下記の補間開始前のエラーコードが返ります。 [C#]の場合は補足説明(1)参照。 ■正常開始 BP_START　バックグラウンドでBP補間処理が正常に開始した ■エラーコード(補間開始前のエラー) BP_CNT_ERR　指定したデータ数が範囲外です BP_ALREADY_EXEC　既にＢＰ補間、あるいは連続補間が実行中です BP_THREAD_ERR　スレッドを起動できませんでした BP_MALLOC_ERR　メモリを確保できませんでした バックグラウンドで補間処理が正常に開始した後は、補間処理終了時に、指定したウィンドウに対して、 WM_BP_ENDメッセージを送信します。WM_BP_ENDのメッセージ受信関数で受け取る第１引数にボード番号を 第２引数に終了ステータスを渡します。 その終了ステータスは、補間が正常終了した場合は BP_END です。 補間実行時にエラーが発生した場合は、下記の補間開始後のエラーコードです。 ■正常終了 BP_END　BP補間処理正常終了 ■エラーコード(補間開始後のエラー) BP_STOP　BP補間が途中で停止した(速度が速く次データのスタックが間に合わなかった) BP_USER_STOP　BP補間実行中にユーザーが中断した BP_DRIVE_ERR　BP補間実行中にMC8043Pでエラーが発生した（RR0にエラー情報がセットされた）

使用例

[VC]

```
    {
        // ２軸ＢＰ補間データ     BP1P,    BP1M,    BP2P,    BP2M
        DATA_2BP Data2Bp[2] = {{0x0000, 0x2BFF, 0xFFD4, 0x0000},
                               {0xF6FE, 0x0000, 0x000F, 0x3FC0}};

        Nmc_WriteReg5(No, 0x04);                              // 補間軸設定。主軸：Ｘ、第２軸：Ｙ
        Ret = Nmc_2BPExec_BG(hWnd, No, Data2Bp, 2, 0x04); // ２軸ＢＰ補間実行。ﾃﾞｰﾀ数2, X,Y軸
        if(Ret == BP_START)    AfxMessageBox("補間開始");  // 戻り値正常(補間開始)
    }

    BEGIN_MESSAGE_MAP(CMC_SAMPLEDlg, CDialog)     // WM_BP_ENDメッセージ受信関数設定
        ON_MESSAGE( WM_BP_END, OnMsg_BP )
    END_MESSAGE_MAP()

    // WM_BP_ENDメッセージ受信関数
    afx_msg LRESULT CMC_SAMPLEDlg::OnMsg_BP(WPARAM BoardNo, LPARAM Status)
    {
        if(Status == BP_END)    AfxMessageBox("補間正常終了"); // 戻り値正常(補間終了)
        return 0;
    }
```

[VB]

```
      Dim Data2Bp(1) As DATA_2BP      '２軸ＢＰ補間データ
      '２軸ＢＰ補間データ設定
      Data2Bp(0).Bp1p = &H0:    Data2Bp(0).Bp1m = &H2BFF
      Data2Bp(0).Bp2p = &HFFD4: Data2Bp(0).Bp2m = &H0
      Data2Bp(1).Bp1p = &HF6FE: Data2Bp(1).Bp1m = &H0
      Data2Bp(1).Bp2p = &HF:    Data2Bp(1).Bp2m = &H3FC0

      Call Nmc_WriteReg5(No, &H4)                                  '補間軸設定。主軸:X, 第2軸:Y
      Ret = Nmc_2BPExec_BG(hWnd, No, Data2Bp(0), 2, &H4, False) '2軸BP補間実行。ﾃﾞｰﾀ数2, X,Y軸
      If Ret = BP_START Then                                       '戻り値正常(補間開始)
          Call MsgBox("補間開始")
      End If
    End Sub
```

ＶＢの場合は次のメッセージ受信関数

```
    'WM_BP_ENDメッセージ受信関数
    Function WindowProc(ByVal hw As Long, ByVal uMsg As Long,
                        ByVal wParam As Long, ByVal lParam As Long) As Long
        If uMsg = WM_BP_END Then           'ＢＰ補間終了メッセージ
            If lParam = BP_END Then        '戻り値正常(補間終了)
                Call MsgBox("補間正常終了")
            End If
        End If

        WindowProc = CallWindowProc(glpPrevWndProc, hw, uMsg, wParam, lParam)

    End Function
```

ＶＢ．ＮＥＴの場合は次のメッセージ受信関数

```
    'WM_BP_ENDメッセージ受信関数
    Protected Overrides Sub WndProc(ByRef m As Message)

        If m.Msg = WM_BP_END Then          'ＢＰ補間終了メッセージ
            If lParam = BP_END Then        '戻り値正常(補間終了)
                Call MsgBox("補間正常終了")
            End If
        End If

        MyBase.WndProc(m)

    End Sub
```

```
[C#]
   DATA_2BP [] Data2Bp = new DATA_2BP[4];  // ２軸ＢＰ補間データ
    // 補間データ設定
    Data2Bp[0].Bp1p = 0xFF30;       // 1111 1111 0011 0000  BP1＋方向       10パルス
    Data2Bp[0].Bp1m = 0;            // 0000 0000 0000 0000  BP1－方向        0パルス
    Data2Bp[0].Bp2p = 0;            // 0000 0000 0000 0000  BP2＋方向        0パルス
    Data2Bp[0].Bp2m = 0x84FF;       // 1000 0100 1111 1111  BP2－方向       10パルス

    IpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y << 2;      // 補間軸

    MC8043P.Nmc_WriteReg5(No, IpAxis);              // 補間軸設定
    MC8043P.Nmc_StartSpd(No, AXIS.ALL, 1);          // 初速度設定
    MC8043P.Nmc_Speed(gBoardNo, AXIS.ALL, 1);       // ドライブ速度設定
    // ２軸ＢＰ補間バックグラウンド実行
    Ret = MC8043P.Nmc_2BPExec_BG((System.IntPtr)parent.Handle, No, Data2Bp,
                                 DataCnt, IpAxis, ContinueFlg);
    if(Ret == Nmc_Status.BP_START)                  // 補間開始正常の場合
```

Ｃ#．ＮＥＴの場合は次のメッセージ受信関数

```
'WM_BP_ENDメッセージ受信関数
protected override void WndProc(ref Message m)
{
    if ( m.Msg == (int)MSG_ID.WM_BP_END )
    {
            if((uint)m.LParam == Nmc_Status.BP_END)  // BP補間処理正常終了

    }
    base.WndProc ( ref m );                         // 元のWndProc呼び出し

 }
```

関数名	機能　及び　内容
Nmc_3BPExec_BG	指定した補間データで３軸ビットパターン補間をバックグラウンドで実行する。 この関数は、補間処理を開始した直後に制御を返し、バックグラウンドで補間を実行します。 指定したウィンドウに対して、補間終了時にWM_BP_ENDメッセージを送信し、終了ステータスを渡します。 **VC**　DWORD　Nmc_3BPExec_BG(HWND User_hWnd, int No, DATA_3BP* pData3Bp, int DataCnt, int IpAxis, BOOL ContinueFlg = FALSE); **VB**　Function Nmc_3BPExec_BG(ByVal User_hWnd As Long, ByVal No As Long, ByRef pData3Bp As DATA_3BP, ByVal DataCnt As Long, ByVal IpAxis As Long, ByVal ContinueFlg As Long) As Long **VB.NET**　Function Nmc_3BPExec_BG(ByVal User_hWnd As Integer, ByVal No As Integer, ByRef pData3Bp As DATA_3BP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal ContinueFlg As Integer) As Integer **C#.NET**　Nmc_Status MC8043P.Nmc_3BPExec_BG(System.IntPtr User_hWnd, int No, DATA_3BP[] pData3Bp, int DataCnt, int IpAxis, bool ContinueFlg); **入力パラメータ** User_hWnd　ユーザーアプリケーションのウィンドウハンドル No　ボード番号（ボード上のロータリースイッチの値(0～9)） pData3Bp　３軸ＢＰ補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_3BPのｱﾄﾞﾚｽ)。 DATA_3BPに実行する補間データをセットし、アドレスを指定する。 DATA_3BPについては補足説明(3)参照。 DataCnt　３軸ＢＰ補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 IpAxis　補間を実行する軸。WR5のD0～D5(軸指定)の設定値と同じ値を指定する。補足説明(4)参照。 ContinueFlg BP補間が途中で停止した（速度が速く次データのスタックが間に合わなかった）場合に 続けるかどうかを設定する。 [VC]　TRUE　：続ける、FALSE　：続けない。省略可能。省略時FALSE。 [VB]　True　：続ける、False　：続けない [C#]　true　：続ける、false　：続けない **戻り値** バックグラウンドで補間処理が正常に開始した場合は BP_START が返ります。 補間処理が開始する前にエラーが発生した場合は、下記の補間開始前のエラーコードが返ります。 [C#]の場合は補足説明(1)参照。 ■正常開始 BP_START　バックグラウンドでBP補間処理が正常に開始した ■エラーコード(補間開始前のエラー) BP_CNT_ERR　指定したデータ数が範囲外です BP_ALREADY_EXEC　既にＢＰ補間、あるいは連続補間が実行中です BP_THREAD_ERR　スレッドを起動できませんでした BP_MALLOC_ERR　メモリを確保できませんでした バックグラウンドで補間処理が正常に開始した後は、補間処理終了時に、指定したウィンドウに対して、 WM_BP_ENDメッセージを送信します。WM_BP_ENDのメッセージ受信関数で受け取る第１引数にボード番号を 第２引数に終了ステータスを渡します。 その終了ステータスは、補間が正常終了した場合は BP_END です。 補間実行時にエラーが発生した場合は、下記の補間開始後のエラーコードです。 ■正常終了 BP_END　BP補間処理正常終了 ■エラーコード(補間開始後のエラー) BP_STOP　BP補間が途中で停止した(速度が速く次データのスタックが間に合わなかった) BP_USER_STOP　BP補間実行中にユーザーが中断した BP_DRIVE_ERR　BP補間実行中にMC8043Pでエラーが発生した（RR0にエラー情報がセットされた）

使用例

[VC]

```
    {
        // ３軸ＢＰ補間データ    BP1P,    BP1M,    BP2P,    BP2M,    BP3P,    BP3M
        DATA_3BP Data3Bp[2] = {{0xFF30,    0,          0, 0x84FF,       0, 0xAC35},
                               {0xAC35,    0,    0xC000, 0x36E7, 0xC000, 0x3F3F}};

        Nmc_WriteReg5(No, 0x24);                              // 補間軸設定。主軸:X, 第2軸:Y, 第3軸:Z
        Ret = Nmc_3BPExec_BG(hWnd, No, Data3Bp, 2, 0x24); // ３軸ＢＰ補間実行。ﾃﾞｰﾀ数2, X,Y,Z軸
        if(Ret == BP_START)   AfxMessageBox("補間開始");  // 戻り値正常(補間開始)
    }

    BEGIN_MESSAGE_MAP(CMC_SAMPLEDlg, CDialog)     // WM_BP_ENDメッセージ受信関数設定
        ON_MESSAGE( WM_BP_END, OnMsg_BP )
    END_MESSAGE_MAP()

    // WM_BP_ENDメッセージ受信関数
    afx_msg LRESULT CMC_SAMPLEDlg::OnMsg_BP(WPARAM BoardNo, LPARAM Status)
    {
        if(Status == BP_END)   AfxMessageBox("補間正常終了"); // 戻り値正常(補間終了)
        return 0;
    }
```

[VB]

```
      Dim Data3Bp(1) As DATA_3BP      '３軸ＢＰ補間データ

      '３軸ＢＰ補間データ設定
      Data3Bp(0).Bp1p = &HFF30: Data3Bp(0).Bp1m = &H0
      Data3Bp(0).Bp2p = &H0:    Data3Bp(0).Bp2m = &H84FF
      Data3Bp(0).Bp3p = &H0:    Data3Bp(0).Bp3m = &HAC35
      Data3Bp(1).Bp1p = &HAC35: Data3Bp(1).Bp1m = &H0
      Data3Bp(1).Bp2p = &HC000: Data3Bp(1).Bp2m = &H36E7
      Data3Bp(1).Bp3p = &HC000: Data3Bp(1).Bp3m = &H3F3F

      Call Nmc_WriteReg5(No, &H24)                            '補間軸設定。主軸:X, 第2軸:Y, 第3軸:Z
      Ret = Nmc_3BPExec_BG(hWnd,No,Data3Bp(0),2,&H24,False) '3軸ＢＰ補間実行。ﾃﾞｰﾀ数2, X,Y,Z軸
      If Ret = BP_START Then                                  '戻り値正常(補間開始)
          Call MsgBox("補間開始")
      End If
    End Sub
```

ＶＢの場合は次のメッセージ受信関数

```
    'WM_BP_ENDメッセージ受信関数
    Function WindowProc(ByVal hw As Long, ByVal uMsg As Long,
                        ByVal wParam As Long, ByVal lParam As Long) As Long
        If uMsg = WM_BP_END Then           'ＢＰ補間終了メッセージ
            If lParam = BP_END Then        '戻り値正常(補間終了)
                Call MsgBox("補間正常終了")
            End If
        End If

        WindowProc = CallWindowProc(glpPrevWndProc, hw, uMsg, wParam, lParam)
    End Function
```

ＶＢ．ＮＥＴの場合は次のメッセージ受信関数

```
    'WM_BP_ENDメッセージ受信関数
    Protected Overrides Sub WndProc(ByRef m As Message)

        If m.Msg = WM_BP_END Then          'ＢＰ補間終了メッセージ
            If lParam = BP_END Then        '戻り値正常(補間終了)
                Call MsgBox("補間正常終了")
            End If
        End If

        MyBase.WndProc(m)
    End Sub
```

```
[C#]
    DATA_3BP [] Data3Bp = new DATA_3BP[4];  // ３軸ＢＰ補間データ
    // 補間データ設定
    Data3Bp[0].Bp1p = 0xFF30;       // 1111 1111 0011 0000  BP1＋方向       10パルス
    Data3Bp[0].Bp1m = 0;            // 0000 0000 0000 0000  BP1－方向        0パルス
    Data3Bp[0].Bp2p = 0;            // 0000 0000 0000 0000  BP2＋方向        0パルス
    Data3Bp[0].Bp2m = 0x84FF;       // 1000 0100 1111 1111  BP2－方向       10パルス
    Data3Bp[0].Bp3p = 0;            // 0000 0000 0000 0000  BP3＋方向        0パルス
    Data3Bp[0].Bp3m = 0xAC35;       // 1010 1100 0011 0101  BP3－方向        8パルス

    IpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y << 2 | IP_AXIS.IP_Z << 4;      // 補間軸

    MC8043P.Nmc_WriteReg5(No, IpAxis);              // 補間軸設定
    MC8043P.Nmc_StartSpd(No, AXIS.ALL, 1);          // 初速度設定
    MC8043P.Nmc_Speed(gBoardNo, AXIS.ALL, 1);       // ドライブ速度設定
    // ３軸ＢＰ補間バックグラウンド実行
    Ret = MC8043P.Nmc_3BPExec_BG((System.IntPtr)parent.Handle, No, Data3Bp,
                                 DataCnt, IpAxis, ContinueFlg);
    if(Ret == Nmc_Status.BP_START)                  // 補間開始正常の場合
```

Ｃ#．ＮＥＴの場合は次のメッセージ受信関数

```
'WM_BP_ENDメッセージ受信関数
protected override void WndProc(ref Message m)
{
     if ( m.Msg == (int)MSG_ID.WM_BP_END )          '
     {
             if((uint)m.LParam == Nmc_Status.BP_END)  // BP補間処理正常終了
     }
     base.WndProc ( ref m );        // 元のWndProc呼び出し

}
```

関数名	機能　及び　内容
Nmc_2CIPExec	指定した補間データで２軸連続補間を実行する。 この関数は、補間処理が終了した後に制御を返します。補間が終了するまで制御が戻らないのでアプリケーションでスレッドを作成し、そのスレッドからコールする事を推奨します。 **VC**　DWORD　Nmc_2CIPExec(int No, DATA_2CIP* pData2Cip, int DataCnt, int lpAxis, BOOL SpdChgFlg = FALSE, BOOL ContinueFlg = FALSE); **VB**　Function Nmc_2CIPExec(ByVal No As Long, ByRef pData2Cip As DATA_2CIP, ByVal DataCnt As Long, ByVal lpAxis As Long, ByVal SpdChgFlg As Long, ByVal ContinueFlg As Long) As Long **VB.NET**　Function Nmc_2CIPExec(ByVal No As Integer, ByRef pData2Cip As DATA_2CIP, ByVal DataCnt As Integer, ByVal lpAxis As Integer, ByVal SpdChgFlg As Integer,ByVal ContinueFlg As Integer) As Integer **C#.NET**　Nmc_Status MC8043P. Nmc_2CIPExec(int No, DATA_2CIP[] pData2Cip, int DataCnt, int lpAxis, bool SpdChgFlg, bool ContinueFlg); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） pData2Cip　２軸連続補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_2CIPのｱﾄﾞﾚｽ)。 DATA_2CIPに実行する補間データをセットし、アドレスを指定する。 DATA_2CIPについては補足説明(3)参照。 DataCnt　２軸連続補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 lpAxis　補間を実行する軸。WR5のD0～D5(軸指定)の設定値と同じ値を指定する。補足説明(4)参照。 SpdChgFlg　補間実行中に速度を変更するかどうかを設定する。変更する場合は補足説明(5)参照。 [VC]　TRUE　：変更する、FALSE　：変更しない。省略可能。省略時FALSE。 [VB]　True　：変更する、False　：変更しない [C#]　true　：変更する、false　：変更しない **変更する選択時**　：DATA_2CIPのSpeedに設定した値を参照します。 Speedに1～8000の値設定・・・設定した速度に変更します。 Speedに0 設定　・・・・・・・速度は変更しません。 **変更しない選択時**：DATA_2CIPのSpeedに設定した値を参照しません。 ContinueFlg 連続補間が途中で停止した（速度が速く次データのセットが間に合わなかった）場合に続けるかどうかを設定する。 [VC]　TRUE　：続ける、FALSE　：続けない。省略可能。省略時FALSE。 [VB]　True　：続ける、False　：続けない [C#]　true　：続ける、false　：続けない **戻り値** 補間処理が正常終了した場合は CIP_END が返ります。 補間処理でエラーが発生した場合は、下記エラーコードが返ります。 [C#]の場合は補足説明(1)参照。 ■正常終了 CIP_END　連続補間処理正常終了 ■エラーコード CIP_CNT_ERR　指定したデータ数が範囲外です CIP_ALREADY_EXEC　既にＢＰ補間、あるいは連続補間が実行中です CIP_CMD_ERR　指定したコマンドが間違っています CIP_STOP　連続補間が途中で停止した(速度が速く次データのセットが間に合わなかった) CIP_USER_STOP　連続補間実行中にユーザーが中断した CIP_DRIVE_ERR　連続補間実行中にMC8043Pでエラーが発生した（RR0にエラー情報がセットされた）

使用例

```
[VC] // ２軸連続補間データ     コマンド 速度 終点１ 終点２ 中心点１ 中心点２
     DATA_2CIP Data2Cip[2]={{CMD_IP_2ST, 0,  4500,     0,       0,       0},  // ２軸直線補間
                            {CMD_IP_CCW, 0,  1500,  1500,       0,    1500}}; // CCW 円弧補間

     Nmc_WriteReg5(No, 0x04);                          // 補間軸設定。主軸：X、第２軸：Y
     Ret = Nmc_2CIPExec(No, Data2Cip, 2, 0x04);        // ２軸連続補間実行。ﾃﾞｰﾀ数2, X,Y軸
     if(Ret == CIP_END)   AfxMessageBox("正常終了"); // 戻り値正常

[VB] Dim Data2Cip(1) As DATA_2CIP          '２軸連続補間データ

     '２軸連続補間データ設定
     Data2Cip(0).Command = CMD_IP_2ST      '２軸直線補間
     Data2Cip(0).EndP1 = 4500
     Data2Cip(0).EndP2 = 0

     Data2Cip(1).Command = CMD_IP_CCW      'ＣＣＷ円弧補間
     Data2Cip(1).EndP1 = 1500
     Data2Cip(1).EndP2 = 1500
     Data2Cip(1).Center1 = 0
     Data2Cip(1).Center2 = 1500

     Call Nmc_WriteReg5(No, &H4)                                  '補間軸設定。主軸：X、第２軸：Y
     Ret = Nmc_2CIPExec(No, Data2Cip(0), 2, &H4, False, False) '2軸連続補間。ﾃﾞｰﾀ数2, X,Y軸
     If Ret = CIP_END Then                                        '戻り値正常
         Call MsgBox("正常終了")
     End If

 [C#]
    DATA_2CIP [] Data2Cip = new DATA_2CIP[4];          // ２軸連続補間データ

    // 補間データ設定
    Data2Cip[0].Command = (ushort)CMD.CMD_IP_CW;       // ＣＷ円弧補間
    Data2Cip[0].EndP1 = 2000;
    Data2Cip[0].EndP2 = 2000;
    Data2Cip[0].Center1 = 2000;
    Data2Cip[0].Center2 = 0;
    Data2Cip[0].Speed = 200;                           // 速度変更する(200)

    IpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y << 2;         // 補間軸

    MC8043P.Nmc_WriteReg5(No, IpAxis);                 // 補間軸設定

    // ２軸連続補間実行
    // この関数は補間が終了するまで制御が戻りません
    Ret = MC8043P.Nmc_2CIPExec(No, Data2Cip, DataCnt, IpAxis, SpdChgFlg, ContinueFlg);
    if(Ret == Nmc_Status.CIP_END)                      // 連続補間処理正常終了
```

関数名	機能　及び　内容
Nmc_3CIPExec	指定した補間データで３軸連続補間を実行する。 この関数は、補間処理が終了した後に制御を返します。補間が終了するまで制御が戻らないのでアプリケーションでスレッドを作成し、そのスレッドからコールする事を推奨します。 **VC**　DWORD　Nmc_3CIPExec(int No, DATA_3CIP* pData3Cip, int DataCnt, int lpAxis, BOOL SpdChgFlg = FALSE, BOOL ContinueFlg = FALSE); **VB**　Function Nmc_3CIPExec(ByVal No As Long, ByRef pData3Cip As DATA_3CIP, ByVal DataCnt As Long, ByVal lpAxis As Long, ByVal SpdChgFlg As Long, ByVal ContinueFlg As Long) As Long **VB.NET**　Function Nmc_3CIPExec(ByVal No As Integer, ByRef pData3Cip As DATA_3CIP, ByVal DataCnt As Integer, ByVal lpAxis As Integer, ByVal SpdChgFlg As Integer,ByVal ContinueFlg As Integer) As Integer **C#.NET**　Nmc_Status MC8043P.Nmc_3CIPExec(int No, DATA_3CIP[] pData3Cip, int DataCnt, int lpAxis, bool SpdChgFlg, bool ContinueFlg); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～9)） pData3Cip　３軸連続補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_3CIPのｱﾄﾞﾚｽ)。 DATA_3CIPに実行する補間データをセットし、アドレスを指定する。 DATA_3CIPについては補足説明(3)参照。 DataCnt　３軸連続補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 lpAxis　補間を実行する軸。WR5のD0～D5(軸指定)の設定値と同じ値を指定する。補足説明(4)参照。 SpdChgFlg　補間実行中に速度を変更するかどうかを設定する。変更する場合は補足説明(5)参照。 [VC]　TRUE　：変更する、FALSE　：変更しない。省略可能。省略時FALSE。 [VB]　True　：変更する、False　：変更しない [C#]　true　：変更する、false　：変更しない **変更する選択時**　：DATA_3CIPのSpeedに設定した値を参照します。 Speedに1～8000の値設定・・・設定した速度に変更します。 Speedに0 設定　・・・・・・・速度は変更しません。 **変更しない選択時**：DATA_3CIPのSpeedに設定した値を参照しません。 ContinueFlg 連続補間が途中で停止した（速度が速く次データのセットが間に合わなかった）場合に続けるかどうかを設定する。 [VC]　TRUE　：続ける、FALSE　：続けない。省略可能。省略時FALSE。 [VB]　True　：続ける、False　：続けない [C#]　true　：続ける、false　：続けない **戻り値** 補間処理が正常終了した場合は CIP_END が返ります。 補間処理でエラーが発生した場合は、下記エラーコードが返ります。 [C#]の場合は補足説明(1)参照。 ■正常終了 CIP_END　連続補間処理正常終了 ■エラーコード CIP_CNT_ERR　指定したデータ数が範囲外です CIP_ALREADY_EXEC　既にＢＰ補間、あるいは連続補間が実行中です CIP_CMD_ERR　指定したコマンドが間違っています CIP_STOP　連続補間が途中で停止した(速度が速く次データのセットが間に合わなかった) CIP_USER_STOP　連続補間実行中にユーザーが中断した CIP_DRIVE_ERR　連続補間実行中にMC8043Pでエラーが発生した（RR0にエラー情報がセットされた）

使用例

```
[VC] DATA_3CIP Data3Cip[2];          // ３軸連続補間データ用

     // ３軸連続補間データ設定
     Data3Cip[0].EndP1 = 1000;
     Data3Cip[0].EndP2 = 2000;
     Data3Cip[0].EndP3 = 3000;
     Data3Cip[0].Speed = 0;
     Data3Cip[1].EndP1 = 2000;
     Data3Cip[1].EndP2 = -1000;
     Data3Cip[1].EndP3 = 3000;
     Data3Cip[1].Speed = 0;

     Nmc_WriteReg5(No, 0x24);                        // 補間軸設定。主軸：X, 第2軸：Y, 第3軸：Z
     Ret = Nmc_3CIPExec(No, Data3Cip, 2, 0x24);      // ３軸連続補間実行。ﾃﾞｰﾀ数2, X,Y,Z軸
     if(Ret == CIP_END)   AfxMessageBox("正常終了"); // 戻り値正常

[VB] Dim Data3Cip(1) As DATA_3CIP    '３軸連続補間データ

     '３軸連続補間データ設定
     Data3Cip(0).EndP1 = 1000
     Data3Cip(0).EndP2 = 2000
     Data3Cip(0).EndP3 = 3000
     Data3Cip(0).Speed = 0

     Data3Cip(1).EndP1 = 2000
     Data3Cip(1).EndP2 = -1000
     Data3Cip(1).EndP3 = 3000
     Data3Cip(1).Speed = 0

     Call Nmc_WriteReg5(No, &H24)                          '補間軸設定。主軸:X, 第2軸:Y, 第3軸:Z
     Ret = Nmc_3CIPExec(No, Data3Cip(0), 2, &H24, False, False) '3軸連続補間。ﾃﾞｰﾀ数2,X,Y,Z軸
     If Ret = CIP_END Then                                 '戻り値正常
         Call MsgBox("正常終了")
     End If

 [C#]
    DATA_3CIP [] Data3Cip = new DATA_3CIP[4];        // ３軸連続補間データ用
    // 補間データ設定
    Data3Cip[0].EndP1 = 1000;
    Data3Cip[0].EndP2 = 2000;
    Data3Cip[0].EndP3 = 3000;
    Data3Cip[0].Speed = 0;

    IpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y << 2 | IP_AXIS.IP_Z << 4;      // 補間軸

    MC8043P.Nmc_WriteReg5(gBoardNo, IpAxis);         // 補間軸設定
    MC8043P.Nmc_StartSpd(gBoardNo, AXIS.ALL, 8000); // 初速度設定
    MC8043P.Nmc_Speed(gBoardNo, AXIS.ALL, 100);      // ドライブ速度設定

    // ３軸連続補間実行
    // この関数は補間が終了するまで制御が戻りません
    Ret = MC8043P.Nmc_3CIPExec(No, Data3Cip, DataCnt, IpAxis, SpdChgFlg, ContinueFlg);
    if(Ret == Nmc_Status.CIP_END)                    // 連続補間処理正常終了
```

関数名	機能　及び　内容
Nmc_2CIPExec_BG	指定した補間データで２軸連続補間をバックグラウンドで実行する。 この関数は、補間処理を開始した直後に制御を返し、バックグラウンドで補間を実行します。 指定したウィンドウに対して、補間終了時にWM_CIP_ENDメッセージを送信し、終了ステータスを渡します。 **VC**　DWORD　Nmc_2CIPExec_BG(HWND User_hWnd, int No, DATA_2CIP* pData2Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE, BOOL ContinueFlg = FALSE); **VB**　Function Nmc_2CIPExec_BG(ByVal User_hWnd As Long, ByVal No As Long, ByRef pData2Cip As DATA_2CIP, ByVal DataCnt As Long, ByVal IpAxis As Long, ByVal SpdChgFlg As Long, ByVal ContinueFlg As Long) As Long **VB.NET**　Function Nmc_2CIPExec_BG(ByVal User_hWnd As Integer, ByVal No As Integer, ByRef pData2Cip As DATA_2CIP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal SpdChgFlg As Integer,ByVal ContinueFlg As Integer) As Integer **C#.NET**　Nmc_Status MC8043P.Nmc_2CIPExec_BG(System.IntPtr User_hWnd, int No, DATA_2CIP[] pData2Cip, int DataCnt, int IpAxis, bool SpdChgFlg, bool ContinueFlg); **入力パラメータ** User_hWnd　ユーザーアプリケーションのウィンドウハンドル No　ボード番号（ボード上のロータリースイッチの値(0～9)） pData2Cip　２軸連続補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_2CIPのｱﾄﾞﾚｽ)。 DATA_2CIPに実行する補間データをセットし、アドレスを指定する。 DATA_2CIPについては補足説明(3)参照。 DataCnt　２軸連続補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 IpAxis　補間を実行する軸。WR5のD0～D5(軸指定)の設定値と同じ値を指定する。補足説明(4)参照。 SpdChgFlg　補間実行中に速度を変更するかどうかを設定する。変更する場合は補足説明(5)参照。 [VC]　TRUE　：変更する、FALSE　：変更しない。省略可能。省略時FALSE。 [VB]　True　：変更する、False　：変更しない [C#]　true　：変更する、false　：変更しない **変更する選択時**　：DATA_2CIPのSpeedに設定した値を参照します。 Speedに1～8000の値設定・・・設定した速度に変更します。 Speedに0 設定　・・・・・・・速度は変更しません。 **変更しない選択時**：DATA_2CIPのSpeedに設定した値を参照しません。 ContinueFlg 連続補間が途中で停止した（速度が速く次データのセットが間に合わなかった）場合に 続けるかどうかを設定する。 [VC]　TRUE　：続ける、FALSE　：続けない。省略可能。省略時FALSE。 [VB]　True　：続ける、False　：続けない [C#]　true　：続ける、false　：続けない **戻り値** バックグラウンドで補間処理が正常に開始した場合は CIP_START が返ります。 補間処理が開始する前にエラーが発生した場合は、下記の補間開始前のエラーコードが返ります。 [C#]の場合は補足説明(1)参照。 ■正常開始 CIP_START　バックグラウンドで連続補間処理が正常に開始した ■エラーコード(補間開始前のエラー) CIP_CNT_ERR　指定したデータ数が範囲外です CIP_ALREADY_EXEC　既にＢＰ補間、あるいは連続補間が実行中です CIP_THREAD_ERR　スレッドを起動できませんでした CIP_MALLOC_ERR　メモリを確保できませんでした CIP_CMD_ERR　指定したコマンドが間違っています バックグラウンドで補間処理が正常に開始した後は、補間処理終了時に、指定したウィンドウに対して、 WM_CIP_ENDメッセージを送信します。WM_CIP_ENDのメッセージ受信関数で受け取る第１引数にボード番号 を、第２引数に終了ステータスを渡します。 その終了ステータスは、補間が正常終了した場合は CIP_END です。 補間実行時にエラーが発生した場合は、下記の補間開始後のエラーコードです。

■正常終了
CIP_END　　連続補間処理正常終了

■エラーコード(補間開始後のエラー)
CIP_STOP　　連続補間が途中で停止した(速度が速く次データのセットが間に合わなかった)
CIP_USER_STOP　　連続補間実行中にユーザーが中断した
CIP_DRIVE_ERR　　連続補間実行中にMC8043Pでエラーが発生した（RR0にエラー情報がセットされた）

使用例

[VC]

```
{
   // ２軸連続補間データ     コマンド 速度 終点１ 終点２ 中心点１ 中心点２
   DATA_2CIP Data2Cip[2]={{CMD_IP_2ST, 0,  4500,     0,      0,      0},  // ２軸直線補間
                          {CMD_IP_CCW, 0,  1500,  1500,      0,   1500}}; // CCW 円弧補間

   Nmc_WriteReg5(No, 0x04);                              // 補間軸設定。主軸：Ｘ、第２軸：Ｙ
   Ret = Nmc_2CIPExec_BG(hWnd, No, Data2Cip, 2, 0x04); // ２軸連続補間実行。ﾃﾞｰﾀ数2, X,Y軸
   if(Ret == CIP_START)   AfxMessageBox("補間開始");   // 戻り値正常(補間開始)
}

BEGIN_MESSAGE_MAP(CMC_SAMPLEDlg, CDialog)     // WM_CIP_ENDメッセージ受信関数設定
   ON_MESSAGE( WM_CIP_END, OnMsg_CIP )
END_MESSAGE_MAP()

// WM_CIP_ENDメッセージ受信関数
afx_msg LRESULT CMC_SAMPLEDlg::OnMsg_CIP(WPARAM BoardNo, LPARAM Status)
{
   if(Status == CIP_END)   AfxMessageBox("補間正常終了"); // 戻り値正常(補間終了)
   return 0;
}
```

[VB]

```
      Dim Data2Cip(1) As DATA_2CIP         '２軸連続補間データ

      '２軸連続補間データ設定
      Data2Cip(0).Command = CMD_IP_2ST     '２軸直線補間
      Data2Cip(0).EndP1 = 4500
      Data2Cip(0).EndP2 = 0

      Data2Cip(1).Command = CMD_IP_CCW     'ＣＣＷ円弧補間
      Data2Cip(1).EndP1 = 1500
      Data2Cip(1).EndP2 = 1500
      Data2Cip(1).Center1 = 0
      Data2Cip(1).Center2 = 1500

      Call Nmc_WriteReg5(No, &H4)                                  '補間軸設定。主軸:X, 第2軸:Y
                                                                   '2軸連続補間。ﾃﾞｰﾀ数2, X,Y軸
      Ret = Nmc_2CIPExec_BG(hWnd, No, Data2Cip(0), 2, &H4, False, False)
      If Ret = CIP_START Then                                      '戻り値正常(補間開始)
          Call MsgBox("補間開始")
      End If
   End Sub
```

ＶＢの場合は次のメッセージ受信関数

```
'WM_CIP_ENDメッセージ受信関数
Function WindowProc(ByVal hw As Long, ByVal uMsg As Long,
                    ByVal wParam As Long, ByVal lParam As Long) As Long
    If uMsg = WM_CIP_END Then           '連続補間終了メッセージ
        If lParam = CIP_END Then        '戻り値正常(補間終了)
            Call MsgBox("補間正常終了")
        End If
    End If
    WindowProc = CallWindowProc(glpPrevWndProc, hw, uMsg, wParam, lParam)
End Function
```

ＶＢ．ＮＥＴの場合は次のメッセージ受信関数

```
'WM_CIP_ENDメッセージ受信関数
Protected Overrides Sub WndProc(ByRef m As Message)
    If m.Msg = WM_CIP_END Then          '連続補間終了メッセージ
        If lParam = CIP_END Then        '戻り値正常(補間終了)
            Call MsgBox("補間正常終了")
        End If
    End If
    MyBase.WndProc(m)
End Sub
```

```
[C#]
   DATA_2CIP [] Data2Cip = new DATA_2CIP[4];        // ２軸連続補間データ

   // 補間データ設定
   Data2Cip[0].Command = (ushort)CMD.CMD_IP_CW;     // ＣＷ円弧補間
   Data2Cip[0].EndP1 = 2000;
   Data2Cip[0].EndP2 = 2000;
   Data2Cip[0].Center1 = 2000;
   Data2Cip[0].Center2 = 0;
   Data2Cip[0].Speed = 200;                         // 速度変更する(200)

   lpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y << 2;       // 補間軸

   MC8043P.Nmc_WriteReg5(No, lpAxis);               // 補間軸設定

   // ２軸連続補間実行
   // バックグラウンドで実行する
   Ret = MC8043P.Nmc_2CIPExec_BG((System.IntPtr)parent.Handle, dNo, Data2Cip,
                                 DataCnt, lpAxis, SpdChgFlg, ContinueFlg);
   if(Ret == Nmc_Status.CIP_START)                  // 連続補間処理正常開始
```

Ｃ#．ＮＥＴの場合は次のメッセージ受信関数

```
'WM_BP_ENDメッセージ受信関数
protected override void WndProc(ref Message m)
{
    if ( m.Msg == (int)MSG_ID.MSG_ID.WM_CIP_END )          '
    {
            if((uint)m.LParam == Nmc_Status.CIP_END)  // 連続補間処理正常終了
    }
    base.WndProc ( ref m );                           // 元のWndProc呼び出し

}
```

関数名	機能　及び　内容
Nmc_3CIPExec_BG	指定した補間データで３軸連続補間をバックグラウンドで実行する。 この関数は、補間処理を開始した直後に制御を返し、バックグラウンドで補間を実行します。 指定したウィンドウに対して、補間終了時にWM_CIP_ENDメッセージを送信し、終了ステータスを渡します。 **VC**　DWORD　Nmc_3CIPExec_BG(HWND User_hWnd, int No, DATA_3CIP* pData3Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE, BOOL ContinueFlg = FALSE); **VB**　Function Nmc_3CIPExec_BG(ByVal User_hWnd As Long, ByVal No As Long, ByRef pData3Cip As DATA_3CIP, ByVal DataCnt As Long, ByVal IpAxis As Long, ByVal SpdChgFlg As Long, ByVal ContinueFlg As Long) As Long **VB.NET**　Function Nmc_3CIPExec_BG(ByVal User_hWnd As Integer, ByVal No As Integer, ByRef pData3Cip As DATA_3CIP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal SpdChgFlg As Integer,ByVal ContinueFlg As Integer) As Integer **C#.NET**　Nmc_Status MC8043P.Nmc_3CIPExec_BG(System.IntPtr User_hWnd, int No, DATA_3CIP[] pData3Cip, int IpAxis, bool SpdChgFlg, bool ContinueFlg); **入力パラメータ** User_hWnd　ユーザーアプリケーションのウィンドウハンドル No　ボード番号（ボード上のロータリースイッチの値(0～9)） pData3Cip　３軸連続補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_3CIPのｱﾄﾞﾚｽ)。 DATA_3CIPに実行する補間データをセットし、アドレスを指定する。 DATA_3CIPについては補足説明(3)参照。 DataCnt　３軸連続補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 IpAxis　補間を実行する軸。WR5のD0～D5(軸指定)の設定値と同じ値を指定する。補足説明(4)参照。 SpdChgFlg　補間実行中に速度を変更するかどうかを設定する。変更する場合は補足説明(5)参照。 [VC]　TRUE　：変更する、FALSE　：変更しない。省略可能。省略時FALSE。 [VB]　True　：変更する、False　：変更しない [C#]　true　：変更する、false　：変更しない **変更する選択時**　：DATA_3CIPのSpeedに設定した値を参照します。 Speedに1～8000の値設定・・・設定した速度に変更します。 Speedに0 設定　・・・・・・・速度は変更しません。 **変更しない選択時**：DATA_3CIPのSpeedに設定した値を参照しません。 ContinueFlg 連続補間が途中で停止した（速度が速く次データのセットが間に合わなかった）場合に 続けるかどうかを設定する。 [VC]　TRUE　：続ける、FALSE　：続けない。省略可能。省略時FALSE。 [VB]　True　：続ける、False　：続けない [C#]　true　：続ける、false　：続けない **戻り値** バックグラウンドで補間処理が正常に開始した場合は CIP_START が返ります。 補間処理が開始する前にエラーが発生した場合は、下記の補間開始前のエラーコードが返ります。 [C#]の場合は補足説明(1)参照。 ■正常開始 CIP_START　バックグラウンドで連続補間処理が正常に開始した ■エラーコード(補間開始前のエラー) CIP_CNT_ERR　指定したデータ数が範囲外です CIP_ALREADY_EXEC　既にＢＰ補間、あるいは連続補間が実行中です CIP_THREAD_ERR　スレッドを起動できませんでした CIP_MALLOC_ERR　メモリを確保できませんでした CIP_CMD_ERR　指定したコマンドが間違っています バックグラウンドで補間処理が正常に開始した後は、補間処理終了時に、指定したウィンドウに対して、WM_CIP_ENDメッセージを送信します。WM_CIP_ENDのメッセージ受信関数で受け取る第１引数にボード番号を、第２引数に終了ステータスを渡します。 その終了ステータスは、補間が正常終了した場合は CIP_END です。 補間実行時にエラーが発生した場合は、下記の補間開始後のエラーコードです。

■正常終了
CIP_END　　連続補間処理正常終了

■エラーコード(補間開始後のエラー)
CIP_STOP　　連続補間が途中で停止した(速度が速く次データのセットが間に合わなかった)
CIP_USER_STOP　　連続補間実行中にユーザーが中断した
CIP_DRIVE_ERR　　連続補間実行中にMC8043Pでエラーが発生した（RR0にエラー情報がセットされた）

使用例

[VC]

```
{
   DATA_3CIP Data3Cip[2];          // ３軸連続補間データ用

   // ３軸連続補間データ設定
   Data3Cip[0].EndP1 = 1000;
   Data3Cip[0].EndP2 = 2000;
   Data3Cip[0].EndP3 = 3000;

   Data3Cip[1].EndP1 = 2000;
   Data3Cip[1].EndP2 = -1000;
   Data3Cip[1].EndP3 = 3000;

   Nmc_WriteReg5(No, 0x24);                        // 補間軸設定。主軸：X, 第2軸：Y, 第3軸：Z
   Ret = Nmc_3CIPExec_BG(hWnd, No, Data3Cip, 2, 0x24); // 3軸連続補間実行。ﾃﾞｰﾀ数2, X,Y,Z軸
   if(Ret == CIP_START)   AfxMessageBox("補間開始");   // 戻り値正常(補間開始)
}

BEGIN_MESSAGE_MAP(CMC_SAMPLEDlg, CDialog)  // WM_CIP_ENDメッセージ受信関数設定
   ON_MESSAGE( WM_CIP_END, OnMsg_CIP )
END_MESSAGE_MAP()

// WM_CIP_ENDメッセージ受信関数
afx_msg LRESULT CMC_SAMPLEDlg::OnMsg_CIP(WPARAM BoardNo, LPARAM Status)
{
   if(Status == CIP_END)   AfxMessageBox("補間正常終了"); // 戻り値正常(補間終了)
   return 0;
}
```

[VB]

```
   Dim Data3Cip(1) As DATA_3CIP    '３軸連続補間データ

   '３軸連続補間データ設定
   Data3Cip(0).EndP1 = 1000
   Data3Cip(0).EndP2 = 2000
   Data3Cip(0).EndP3 = 3000

   Data3Cip(1).EndP1 = 2000
   Data3Cip(1).EndP2 = -1000
   Data3Cip(1).EndP3 = 3000

   Call Nmc_WriteReg5(No, &H24)                        '補間軸設定。主軸:X, 第2軸:Y, 第3軸:Z
                                                       '3軸連続補間。ﾃﾞｰﾀ数2, X,Y,Z軸
   Ret = Nmc_3CIPExec_BG(hWnd, No, Data3Cip(0), 2, &H24, False, False)
   If Ret = CIP_START Then                             '戻り値正常(補間開始)
       Call MsgBox("補間開始")
   End If
End Sub
```

ＶＢの場合は次のメッセージ受信関数

```
'WM_CIP_ENDメッセージ受信関数
Function WindowProc(ByVal hw As Long, ByVal uMsg As Long,
                    ByVal wParam As Long, ByVal lParam As Long) As Long

    If uMsg = WM_CIP_END Then           '連続補間終了メッセージ
        If lParam = CIP_END Then        '戻り値正常(補間終了)
            Call MsgBox("補間正常終了")
        End If
    End If

    WindowProc = CallWindowProc(glpPrevWndProc, hw, uMsg, wParam, lParam)

End Function
```

ＶＢ．ＮＥＴの場合は次のメッセージ受信関数

```
'WM_CIP_ENDメッセージ受信関数
Protected Overrides Sub WndProc(ByRef m As Message)

    If m.Msg = WM_CIP_END Then          '連続補間終了メッセージ
        If lParam = CIP_END Then        '戻り値正常(補間終了)
            Call MsgBox("補間正常終了")
        End If
    End If

    MyBase.WndProc(m)

End Sub
```

```
[C#]
   DATA_3CIP [] Data3Cip = new DATA_3CIP[4];        // ３軸連続補間データ用
   // 補間データ設定
   Data3Cip[0].EndP1 = 1000;
   Data3Cip[0].EndP2 = 2000;
   Data3Cip[0].EndP3 = 3000;
   Data3Cip[0].Speed = 0;

   lpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y << 2 | IP_AXIS.IP_Z << 4;      // 補間軸

   MC8043P.Nmc_WriteReg5(gBoardNo, lpAxis);          // 補間軸設定
   MC8043P.Nmc_StartSpd(gBoardNo, AXIS.ALL, 8000);  // 初速度設定
   MC8043P.Nmc_Speed(gBoardNo, AXIS.ALL, 100);       // ドライブ速度設定

   // ３軸連続補間実行
   // バックグラウンドで実行する
   Ret = MC8043P.Nmc_3CIPExec_BG((System.IntPtr)parent.Handle, gBoardNo, Data3Cip,
                                 DataCnt, lpAxis, SpdChgFlg, ContinueFlg);
   if(Ret == Nmc_Status.CIP_START)                   // 連続補間処理正常開始
```

Ｃ#．ＮＥＴの場合は次のメッセージ受信関数

```
'WM_BP_ENDメッセージ受信関数
protected override void WndProc(ref Message m)
{
    if ( m.Msg == (int)MSG_ID.MSG_ID.WM_CIP_END )          '
    {
            if((uint)m.LParam == Nmc_Status.CIP_END)  // 連続補間処理正常終了
    }
    base.WndProc ( ref m );        // 元のWndProc呼び出し

}
```

関数名	機能　及び　内容

Nmc_IPStop	補間処理を中断する。
	補間ドライブを即停止し、Nmc_xxxの補間関数で実行中の補間処理を終了します。
	
	Nmc_IPStopを使用して補間処理を中断した場合、各補間関数の戻り値は下記のエラーコードです。
	◆ＢＰ補間 ： BP_USER_STOP
	◆連続補間 ： CIP_USER_STOP
	
	VC BOOL Nmc_IPStop(int No);
	VB Function Nmc_IPStop(ByVal No As Long) As Long
	VB.NET Function Nmc_IPStop(ByVal No As Integer) As Integer
	C#.NET bool MC8043P.Nmc_IPStop(int No);
	
	入力パラメータ
	No ボード番号（ボード上のロータリースイッチの値(0～9)）
	
	戻り値
	[VC] 補間処理の中断に成功するとTRUE、失敗するとFALSE
	[VB] 補間処理の中断に成功すると０以外、失敗すると０
	[C#] 補間処理の中断に成功するとtrue、失敗するとfalse
	
	使用例
	[VC] Nmc_IPStop(No); // 補間処理を中断する
	[VB] Call Nmc_IPStop(No)
	[C#] MC8043P.Nmc_IPStop(No);

■補足説明

(1) 各定義は、下記のファイルで行っています。
VC・・・・MC8043P_DLL.H
VB・・・・MC8043P_DLL.bas
VB.NET・・MC8043P_DLL.vb
C#.NET・・入力支援により各定義を参照、引用できる。

ＶＣ、Ｃ＃．ＮＥＴの定義内容を以下に示します。
①アドレス定義
[VC]

```
#define MCX314_WR0              0x0000  // ＷＲ０のアドレス
#define MCX314_WR1              0x0001  // ＷＲ１    〃
#define MCX314_WR2              0x0002  // ＷＲ２    〃
#define MCX314_WR3              0x0003  // ＷＲ３    〃
#define MCX314_WR4              0x0004  // ＷＲ４    〃
#define MCX314_WR5              0x0005  // ＷＲ５    〃
#define MCX314_WR6              0x0006  // ＷＲ６    〃
#define MCX314_WR7              0x0007  // ＷＲ７    〃

#define MCX314_RR0              0x0000  // ＲＲ０のアドレス
#define MCX314_RR1              0x0001  // ＲＲ１    〃
#define MCX314_RR2              0x0002  // ＲＲ２    〃
#define MCX314_RR3              0x0003  // ＲＲ３    〃
#define MCX314_RR4              0x0004  // ＲＲ４    〃
#define MCX314_RR5              0x0005  // ＲＲ５    〃
#define MCX314_RR6              0x0006  // ＲＲ６    〃
#define MCX314_RR7              0x0007  // ＲＲ７    〃
```

[C#]

```
public enum REG_MCX314 : int
{
        // MCX314アドレス
        WR0     =0x0000,
        WR1     =0x0001,
        WR2     =0x0002,
        WR3     =0x0003,
        WR4     =0x0004,
        WR5     =0x0005,
        WR6     =0x0006,
        WR7     =0x0007,

        RR0     =0x0000,
        RR1     =0x0001,
        RR2     =0x0002,
        RR3     =0x0003,
        RR4     =0x0004,
        RR5     =0x0005,
        RR6     =0x0006,
        RR7     =0x0007,
}
```

（使用例）MCX314のWR0の場合は　　REG_MCX314.WR0

```
public enum REG_MC8041P : ushort
{
        // レジスタ定義
        WR0             =1,
        WR1             =2,
        WR2             =3,
        WR3             =4,
        WR4             =5,
        WR5             =6,
        WR6             =7,
        WR7             =8,

        RR0             =9,
        RR1             =10,
        RR2             =11,
        RR3             =12,
        RR4             =13,
        RR5             =14,
        RR6             =15,
        RR7             =16,
}
```

（使用例）MC8041PのWR0の場合は　　REG_MC8041P.WR0

②軸定義

[VC]

```
#define AXIS_ALL                0xF     // 全軸
#define AXIS_X                  0x1     // Ｘ軸
#define AXIS_Y                  0x2     // Ｙ軸
#define AXIS_Z                  0x4     // Ｚ軸
#define AXIS_U                  0x8     // Ｕ軸
#define AXIS_NONE               0       // 軸指定無し
```

[C#]

```
public enum AXIS : int
{
        // 軸定義
        ALL                     =0xF,
        X                       =0x1,
        Y                       =0x2,
        Z                       =0x4,
        U                       =0x8,
        NONE                    =0                      // 軸指定無し
}
```

（使用例）全軸の場合は　　AXIS.ALL

③コマンド定義

[VC]

```
// ドライブ命令
#define CMD_F_DRV_P             0x20    // ＋方向定量パルスドライブ
#define CMD_F_DRV_M             0x21    // －方向定量パルスドライブ
#define CMD_C_DRV_P             0x22    // ＋方向連続パルスドライブ
#define CMD_C_DRV_M             0x23    // －方向連続パルスドライブ
#define CMD_START_HOLD          0x24    // ドライブ開始ホールド
#define CMD_START_FREE          0x25    // ドライブ開始フリー／終了ステータスクリア
#define CMD_STP_STS_CLR         0x25    // ドライブ開始フリー／終了ステータスクリア
#define CMD_STOP_DEC            0x26    // ドライブ減速停止
#define CMD_STOP_SUDDEN         0x27    // ドライブ即停止
```

```
// 補間命令
#define CMD_IP_2ST              0x30    // ２軸直線補間ドライブ
#define CMD_IP_3ST              0x31    // ３軸直線補間ドライブ
#define CMD_IP_CW               0x32    // ＣW円弧補間ドライブ
#define CMD_IP_CCW              0x33    // ＣＣW円弧補間ドライブ
#define CMD_IP_2BP              0x34    // ２軸ビットパターン補間ドライブ
#define CMD_IP_3BP              0x35    // ３軸ビットパターン補間ドライブ
#define CMD_BP_ENABLED          0x36    // ＢＰレジスタ書き込み可
#define CMD_BP_DISABLED         0x37    // ＢＰレジスタ書き込み不可
#define CMD_BP_STACK            0x38    // ＢＰデータスタック
#define CMD_BP_CLR              0x39    // ＢＰデータクリア
#define CMD_IP_1STEP            0x3A    // 補間シングルステップ
#define CMD_IP_DEC_VALID        0x3B    // 減速有効
#define CMD_IP_DEC_INVALID      0x3C    // 減速無効
#define CMD_IP_INTRPT_CLR       0x3D    // 補間割り込みクリア

// その他の命令
#define CMD_HOME_EXEC           0x62    // 自動原点出し実行
#define CMD_DEVCTR_CLR          0x63    // 偏差カウンタクリアパルス出力
#define CMD_SYNC_ACTIVE         0x65    // 同期動作起動
#define CMD_NOP                 0x0F    // ＮＯＰ（軸切り換え用）
```

[C#]

```
public enum CMD : int
{
        // データ書き込み命令
        CMD_Range               = 0x00,     // レンジ設定
        CMD_Jerk                = 0x01,     // 加速度増加率設定
        CMD_Acc                 = 0x02,     // 加速度設定
        CMD_Dec                 = 0x03,     // 減速度設定
        CMD_StartSpd            = 0x04,     // 初速度設定
        CMD_Speed               = 0x05,     // ドライブ速度設定
        CMD_Pulse               = 0x06,     // 出力パルス数／補間終点設定
        CMD_DecP                = 0x07,     // マニュアル減速点設定
        CMD_Center              = 0x08,     // 円弧中心点設定
        CMD_Lp                  = 0x09,     // 論理位置カウンタ設定
        CMD_Ep                  = 0x0A,     // 実位置カウンタ設定
        CMD_CompP               = 0x0B,     // ＣＯＭＰ＋レジスタ設定
        CMD_CompM               = 0x0C,     // ＣＯＭＰ－レジスタ設定
        CMD_AccOfst             = 0x0D,     // 加速カウンタオフセット設定
        CMD_DJerk               = 0x0E,     // 減速度増加率設定

        CMD_ExpMode             = 0x60,     // 拡張モード設定
        CMD_HomeSpd             = 0x61,     // 原点検出速度設定
        CMD_SyncMode            = 0x64,     // 同期動作モード設定

        // データ読み出し命令
        CMD_ReadLp              = 0x10,     // 論理位置カウンタ読み出し
        CMD_ReadEp              = 0x11,     // 実位置カウンタ読み出し
        CMD_ReadSpeed           = 0x12,     // 現在ドライブ速度読み出し
        CMD_ReadAccDec          = 0x13,     // 現在加/減速度読み出し
        CMD_ReadSyncBuff        = 0x14,     // 同期バッファレジスタ読み出し

        // ドライブ命令
        CMD_F_DRV_P             = 0x20,     // ＋方向定量パルスドライブ
        CMD_F_DRV_M             = 0x21,     // －方向定量パルスドライブ
        CMD_C_DRV_P             = 0x22,     // ＋方向連続パルスドライブ
        CMD_C_DRV_M             = 0x23,     // －方向連続パルスドライブ
        CMD_START_HOLD          = 0x24,     // ドライブ開始ホールド
        CMD_START_FREE          = 0x25,     // ドライブ開始フリー／終了ステータスクリア
        CMD_STP_STS_CLR         = 0x25,     // ドライブ開始フリー／終了ステータスクリア
        CMD_STOP_DEC            = 0x26,     // ドライブ減速停止
        CMD_STOP_SUDDEN         = 0x27,     // ドライブ即停止
```

```
        // 補間命令
        CMD_IP_2ST              = 0x30,     // ２軸直線補間ドライブ
        CMD_IP_3ST              = 0x31,     // ３軸直線補間ドライブ
        CMD_IP_CW               = 0x32,     // ＣＷ円弧補間ドライブ
        CMD_IP_CCW              = 0x33,     // ＣＣＷ円弧補間ドライブ
        CMD_IP_2BP              = 0x34,     // ２軸ビットパターン補間ドライブ
        CMD_IP_3BP              = 0x35,     // ３軸ビットパターン補間ドライブ
        CMD_BP_ENABLED          = 0x36,     // ＢＰレジスタ書き込み可
        CMD_BP_DISABLED         = 0x37,     // ＢＰレジスタ書き込み不可
        CMD_BP_STACK            = 0x38,     // ＢＰデータスタック
        CMD_BP_CLR              = 0x39,     // ＢＰデータクリア
        CMD_IP_1STEP            = 0x3A,     // 補間シングルステップ
        CMD_IP_DEC_VALID        = 0x3B,     // 減速有効
        CMD_IP_DEC_INVALID      = 0x3C,     // 減速無効
        CMD_IP_INTRPT_CLR       = 0x3D,     // 補間割り込みクリア

        // その他の命令
        CMD_HOME_EXEC           = 0x62,     // 自動原点出し実行
        CMD_DEVCTR_CLR          = 0x63,     // 偏差カウンタクリアパルス出力
        CMD_SYNC_ACTIVE         = 0x65,     // 同期動作起動
        CMD_NOP                 = 0x0F,     // ＮＯＰ（軸切り換え用）
}
```

（使用例）２軸直線補間ドライブを指定する場合は　　CMD.CMD_IP_2ST

④補間終了メッセージ、終了ステータス

[VC]

```
// 補間終了メッセージ
#define WM_BP_END               (WM_USER + 1)   // ＢＰ補間終了メッセージ
#define WM_CIP_END              (WM_USER + 2)   // 連続補間終了メッセージ

//***** BP補間　終了ステータス *****
// ■正常
#define BP_START                0x101   // バックグラウンドでBP補間を開始した
#define BP_END                  0x102   // BP補間正常終了

// ■補間開始前のエラー
#define BP_CNT_ERR              0x111   // 指定されたデータ数が範囲外
#define BP_ALREADY_EXEC         0x112   // 既にＢＰ補間、あるいは連続補間が実行中
#define BP_THREAD_ERR           0x113   // スレッドを起動できなかった
#define BP_MALLOC_ERR           0x114   // メモリを確保できなかった

// ■補間実行中のエラー
#define BP_STOP                 0x121   // BP補間が途中で停止した（速度が速く次データのｽﾀｯｸが間に合わなかった場合）
#define BP_USER_STOP            0x122   // BP補間実行中にユーザーが中断した
#define BP_DRIVE_ERR            0x123   // BP補間実行中にボードでエラー発生（RR0にエラー情報がセットされた）

//***** 連続補間　終了ステータス *****
// ■正常
#define CIP_START               0x201   // バックグラウンドで連続補間を開始した
#define CIP_END                 0x202   // 連続補間正常終了

// ■補間開始前のエラー
#define CIP_CNT_ERR             0x211   // 指定されたデータ数が範囲外
#define CIP_ALREADY_EXEC        0x212   // 既にＢＰ補間、あるいは連続補間が実行中
#define CIP_THREAD_ERR          0x213   // スレッドを起動できなかった
#define CIP_MALLOC_ERR          0x214   // メモリを確保できなかった
#define CIP_CMD_ERR             0x215   // コマンドエラー（ユーザーが指定したコマンドが間違っている）

// ■補間実行中のエラー
#define CIP_STOP                0x221   // 連続補間が途中で停止した（速度が速く次データのｾｯﾄが間に合わなかった場合）
#define CIP_USER_STOP           0x222   // 連続補間実行中にユーザーが中断した
#define CIP_DRIVE_ERR           0x223   // 連続補間実行中にボードでエラー発生（RR0にｴﾗｰ情報がセットされた）
```

[C#]

```
public enum Nmc_Status : uint
{
    //***** BP補間　終了ステータス *****
    // ■正常
    BP_START                = 0x101,    // バックグラウンドでBP補間を開始した
    BP_END                  = 0x102,    // BP補間正常終了

    // ■補間開始前のエラー
    BP_CNT_ERR              = 0x111,    // 指定されたデータ数が範囲外
    BP_ALREADY_EXEC         = 0x112,    // 既にＢＰ補間、あるいは連続補間が実行中
    BP_THREAD_ERR           = 0x113,    // スレッドを起動できなかった
    BP_MALLOC_ERR           = 0x114,    // メモリを確保できなかった

    // ■補間実行中のエラー
    BP_STOP                 = 0x121,    // BP補間が途中で停止した
                                        //（速度が速く次データのスタックが間に合わなかった場合）
    BP_USER_STOP            = 0x122,    // BP補間実行中にユーザーが中断した
    BP_DRIVE_ERR            = 0x123,    // BP補間実行中にボードでエラー発生（RR0にエラー情報がセットされた）

    //***** 連続補間　終了ステータス *****
    // ■正常
    CIP_START               = 0x201,    // バックグラウンドで連続補間を開始した
    CIP_END                 = 0x202,    // 連続補間正常終了

    // ■補間開始前のエラー
    CIP_CNT_ERR             = 0x211,    // 指定されたデータ数が範囲外
    CIP_ALREADY_EXEC        = 0x212,    // 既にＢＰ補間、あるいは連続補間が実行中
    CIP_THREAD_ERR          = 0x213,    // スレッドを起動できなかった
    CIP_MALLOC_ERR          = 0x214,    // メモリを確保できなかった
    CIP_CMD_ERR             = 0x215,    // コマンドエラー（ユーザーが指定したコマンドが間違っている）

    // ■補間実行中のエラー
    CIP_STOP                = 0x221,    // 連続補間が途中で停止した
                                        //（速度が速く次データのセットが間に合わなかった場合）
    CIP_USER_STOP           = 0x222,    // 連続補間実行中にユーザーが中断した
    CIP_DRIVE_ERR           = 0x223,    // 連続補間実行中にボードでエラー発生（RR0にエラー情報がセットされた）
}
```

（使用例）バックグラウンドでBP補間を開始した場合は　　Nmc_Status.BP_START

⑤定数定義

[C#]

```
public enum CONST : int
{
        MAX_BOARD_MC8043P   =10,        // MC8043Pデバイスドライバが同時に認識するボードの最大数(10枚)
        MAX_BOARD_MC8041P   =4,         // MC8041Pデバイスドライバが同時に認識するボードの最大数(4枚まで)
}
```

（使用例）MC8043Pボードの最大数を指定する場合は　　CONST.MAX_BOARD_MC8043P

⑥補間終了メッセージＩＤ

[C#]

```
public enum MSG_ID : int
{       // 補間終了メッセージ
        WM_USER                         =0x0400,
        WM_BP_END                       =WM_USER+1,             // (WM_USER + 1) ＢＰ補間終了メッセージ
        WM_CIP_END                      =WM_USER+2              // (WM_USER + 2) 連続補間終了メッセージ

}
```

（使用例）バックグラウンドから受信するＢＰ補間終了メッセージを指定する場合は　　MSG_ID.WM_BP_END

(2) 軸指定の方法は、下記の通りです。

軸	VC	C#
X	AXIS_X	AXIS.X
Y	AXIS_Y	AXIS.Y
Z	AXIS_Z	AXIS.Z
U	AXIS_U	AXIS.U
全軸	AXIS_ALL	AXIS.ALL

①１軸指定の場合
　　Ｘ軸、Ｙ軸、Ｚ軸のいずれかを指定します。

　　(使用例)　Ｘ軸にドライブ速度 1000 を設定する
　　[VC]　Nmc_Speed(No, AXIS_X, 1000);
　　[VB]　Call Nmc_Speed(No, AXIS_X, 1000)
　　[C#]　MC8043P.Nmc_Speed(No, AXIS.X, 1000);

②２軸指定の場合
　　ビットＯＲ演算子を使用します。
　　例えばＸ軸とＹ軸を一度に指定する場合、
　　[VC]・・・AXIS_X | AXIS_Y を指定します。
　　[VB]・・・AXIS_X Or AXIS_Y を指定します。
　　[C#]・・・AXIS.X | AXIS.Y を指定します。
　　(使用例)　Ｘ,Ｙ軸にドライブ速度 1000 を設定する
　　[VC]　Nmc_Speed(No, AXIS_X | AXIS_Y, 1000);
　　[VB]　Call Nmc_Speed(No, AXIS_X Or AXIS_Y, 1000)
　　[C#]　MC8043P.Nmc_Speed(No, AXIS.X | AXIS.Y, 1000);

③３軸指定の場合
　　ビットＯＲ演算子を使用します。
　　例えばＸ軸とＹ軸とＺ軸を一度に指定する場合、
　　[VC]・・・AXIS_X | AXIS_Y | AXIS_Z を指定します。
　　[VB]・・・AXIS_X Or AXIS_Y Or AXIS_Z を指定します。
　　[C#]・・・AXIS.X | AXIS.Y| AXIS.Z　を指定します。
　　(使用例)　Ｘ,Ｙ,Ｚ軸にドライブ速度 1000 を設定する
　　[VC]　Nmc_Speed(No, AXIS_X | AXIS_Y | AXIS_Z, 1000);
　　[VB]　Call Nmc_Speed(No, AXIS_X Or AXIS_Y Or AXIS_Z, 1000)
　　[C#]　 MC8043P.Nmc_Speed(No, AXIS.X | AXIS.Y| AXIS.Z, 1000);

④全軸指定の場合
　　全軸 を指定します。

　　(使用例)　全軸にドライブ速度 1000 を設定する
　　[VC]　Nmc_Speed(No, AXIS_ALL, 1000);
　　[VB]　Call Nmc_Speed(No, AXIS_ALL, 1000)
　　[C#]　 MC8043P.Nmc_Speed(No, AXIS.ALL, 1000);

(3) 補間関数で使用する構造体（VBではユーザー定義型）の定義は、下記の通りです。

①ＶＣ

```
// ２軸ＢＰ補間
typedef struct _DATA_2BP
{
        USHORT Bp1p;            // BP1Pデータ
        USHORT Bp1m;            // BP1Mデータ
        USHORT Bp2p;            // BP2Pデータ
        USHORT Bp2m;            // BP2Mデータ
} DATA_2BP;
```

```
// ３軸ＢＰ補間
typedef struct _DATA_3BP
{
        USHORT Bp1p;            // BP1Pデータ
        USHORT Bp1m;            // BP1Mデータ
        USHORT Bp2p;            // BP2Pデータ
        USHORT Bp2m;            // BP2Mデータ
        USHORT Bp3p;            // BP3Pデータ
        USHORT Bp3m;            // BP3Mデータ
} DATA_3BP;

// ２軸連続補間
typedef struct _DATA_2CIP
{
        USHORT  Command;        // 命令番号（CMD_IP_2ST, CMD_IP_CW, CMD_IP_CCWのいずれかをセットする）
        USHORT  Speed;          // 速度（速度を変更する場合は１～８０００、変更しない場合は０をセットする）
        long    EndP1;          // 終点（第１軸）
        long    EndP2;          // 終点（第２軸）
        long    Center1;        // 円弧中心点（第１軸）
        long    Center2;        // 円弧中心点（第２軸）
} DATA_2CIP;                    // 注：第１軸、第２軸は、WR5で指定する

// ３軸連続補間
typedef struct _DATA_3CIP
{
        long    EndP1;          // 終点（第１軸）
        long    EndP2;          // 終点（第２軸）
        long    EndP3;          // 終点（第３軸）
        USHORT  Speed;          // 速度（速度を変更する場合は１～８０００、変更しない場合は０をセットする）
} DATA_3CIP;                    // 注：第１軸、第２軸、第３軸は、WR5で指定する
```

②ＶＢ

```
' ２軸ＢＰ補間
Type DATA_2BP
    Bp1p As Integer         ' BP1Pデータ
    Bp1m As Integer         ' BP1Mデータ
    Bp2p As Integer         ' BP2Pデータ
    Bp2m As Integer         ' BP2Mデータ
End Type

' ３軸ＢＰ補間
Type DATA_3BP
    Bp1p As Integer         ' BP1Pデータ
    Bp1m As Integer         ' BP1Mデータ
    Bp2p As Integer         ' BP2Pデータ
    Bp2m As Integer         ' BP2Mデータ
    Bp3p As Integer         ' BP3Pデータ
    Bp3m As Integer         ' BP3Mデータ
End Type

' ２軸連続補間
Type DATA_2CIP
    Command As Integer      ' 命令番号（CMD_IP_2ST, CMD_IP_CW, CMD_IP_CCWのいずれかをセットする）
    Speed As Integer        ' 速度（速度を変更する場合は１～８０００、変更しない場合は０をセットする）
    EndP1 As Long           ' 終点（第１軸）
    EndP2 As Long           ' 終点（第２軸）
    Center1 As Long         ' 円弧中心点（第１軸）
    Center2 As Long         ' 円弧中心点（第２軸）
End Type                    ' 注：第１軸、第２軸は、WR5で指定する

' ３軸連続補間
Type DATA_3CIP
    EndP1 As Long           ' 終点（第１軸）
    EndP2 As Long           ' 終点（第２軸）
    EndP3 As Long           ' 終点（第３軸）
    Speed As Integer        ' 速度（速度を変更する場合は１～８０００、変更しない場合は０をセットする）
End Type                    ' 注：第１軸、第２軸、第３軸は、WR5で指定する
```

③ＶＢ．ＮＥＴ

```
’２軸ＢＰ補間
Structure DATA_2BP
    Dim Bp1p As Short       ’BP1Pデータ
    Dim Bp1m As Short       ’BP1Mデータ
    Dim Bp2p As Short       ’BP2Pデータ
    Dim Bp2m As Short       ’BP2Mデータ
End Structure

’３軸ＢＰ補間
Structure DATA_3BP
    Dim Bp1p As Short       ’BP1Pデータ
    Dim Bp1m As Short       ’BP1Mデータ
    Dim Bp2p As Short       ’BP2Pデータ
    Dim Bp2m As Short       ’BP2Mデータ
    Dim Bp3p As Short       ’BP3Pデータ
    Dim Bp3m As Short       ’BP3Mデータ
End Structure

’２軸連続補間
Structure DATA_2CIP
    Dim Cmd As Short        ’命令番号（CMD_IP_2ST, CMD_IP_CW, CMD_IP_CCWのいずれかをセットする）
    Dim Speed As Short      ’速度（速度を変更する場合は１～８０００、変更しない場合は０をセットする）
    Dim EndP1 As Integer    ’終点（第１軸）
    Dim EndP2 As Integer    ’終点（第２軸）
    Dim Center1 As Integer  ’円弧中心点（第１軸）
    Dim Center2 As Integer  ’円弧中心点（第２軸）
End Structure               ’注：第１軸、第２軸は、WR5で指定する

’３軸連続補間
Structure DATA_3CIP
    Dim EndP1 As Integer    ’終点（第１軸）
    Dim EndP2 As Integer    ’終点（第２軸）
    Dim EndP3 As Integer    ’終点（第３軸）
    Dim Speed As Short      ’速度（速度を変更する場合は１～８０００、変更しない場合は０をセットする）
End Structure               ’注：第１軸、第２軸、第３軸は、WR5で指定する
```

④Ｃ＃．ＮＥＴ

```
// ２軸ＢＰ補間
[StructLayout(LayoutKind.Sequential)]
public struct DATA_2BP
{
        public  DATA_2BP(ushort bp1p,ushort bp1m,ushort bp2p,ushort bp2m)
        {
        this.Bp1p = bp1p;
        this.Bp1m = bp1m;
        this.Bp2p = bp2p;
        this.Bp2m = bp2m;
        }
        public  ushort Bp1p;            // BP1Pデータ
        public  ushort Bp1m;            // BP1Mデータ
        public  ushort Bp2p;            // BP2Pデータ
        public  ushort Bp2m;            // BP2Mデータ
}
```

```
// ３軸ＢＰ補間
[StructLayout(LayoutKind.Sequential)]
public struct DATA_3BP
{
        public  DATA_3BP(ushort bp1p,ushort bp1m,ushort bp2p,ushort bp2m,ushort bp3p,ushort bp3m)
        {
                this.Bp1p = bp1p;
                this.Bp1m = bp1m;
                this.Bp2p = bp2p;
                this.Bp2m = bp2m;
                this.Bp3p = bp3p;
                this.Bp3m = bp3m;
        }
        public  ushort Bp1p;            // BP1Pデータ
        public  ushort Bp1m;            // BP1Mデータ
        public  ushort Bp2p;            // BP2Pデータ
        public  ushort Bp2m;            // BP2Mデータ
        public  ushort Bp3p;            // BP3Pデータ
        public  ushort Bp3m;            // BP3Mデータ
}

// ２軸連続補間
[StructLayout(LayoutKind.Sequential)]
public struct DATA_2CIP
{
    public      DATA_2CIP(ushort command,ushort speed,int endp1,int endp2,int center1,int center2)
    {
        this.Command = command;
        this.Speed = speed;
        this.EndP1 = endp1;
        this.EndP2 = endp2;
        this.Center1 = center1;
        this.Center2 = center2;
    }
    public    ushort     Command;       // 命令番号（CMD_IP_2ST, CMD_IP_CW, CMD_IP_CCWのいずれかをセットする）
    public    ushort     Speed;         // 速度（指定する場合は１～８０００、指定しない場合は０をセットする）
    public    int        EndP1;         // 終点（第１軸）
    public    int        EndP2;         // 終点（第２軸）
    public    int        Center1;       // 円弧中心点（第１軸）
    public    int        Center2;       // 円弧中心点（第２軸）
}                       // 注：第１軸、第２軸は、WR5で指定する
```

```
// ３軸連続補間
[StructLayout(LayoutKind.Sequential)]
public struct DATA_3CIP
{
        public  DATA_3CIP(int endp1,int endp2,int endp3,ushort speed)
        {
                this.EndP1 = endp1;
                this.EndP2 = endp2;
                this.EndP3 = endp3;
                this.Speed = speed;
        }
        public  int             EndP1;          // 終点（第１軸）
        public  int             EndP2;          // 終点（第２軸）
        public  int             EndP3;          // 終点（第３軸）
        public  ushort  Speed;          // 速度（指定する場合は１～８０００、指定しない場合は０をセットする）
}                       // 注：第１軸、第２軸、第３軸は、WR5で指定する
```

構造体の使用例を以下に示します。

例１．定義、初期化が別の場合

```
DATA_2BP [] Data2Bp = new DATA_2BP[4];  // ２軸ＢＰ補間データ

// 補間データ設定
Data2Bp[0].Bp1p = 0xFF30;       // 1111 1111 0011 0000  BP1＋方向       10パルス
Data2Bp[0].Bp1m = 0;            // 0000 0000 0000 0000  BP1－方向        0パルス
Data2Bp[0].Bp2p = 0;            // 0000 0000 0000 0000  BP2＋方向        0パルス
Data2Bp[0].Bp2m = 0x84FF;       // 1000 0100 1111 1111  BP2－方向       10パルス

Data2Bp[1].Bp1p = 0xAC35;       // 1010 1100 0011 0101  BP1＋方向        8パルス
Data2Bp[1].Bp1m = 0;            // 0000 0000 0000 0000  BP1－方向        0パルス
Data2Bp[1].Bp2p = 0xC000;       // 1100 0000 0000 0000  BP2＋方向        2パルス
Data2Bp[1].Bp2m = 0x36E7;       // 0011 0110 1110 0111  BP2－方向       10パルス

Data2Bp[2].Bp1p = 0x3F3F;       // 0011 1111 0011 1111  BP1＋方向       12パルス
Data2Bp[2].Bp1m = 0xC000;       // 1100 0000 0000 0000  BP1－方向        2パルス
Data2Bp[2].Bp2p = 0xFBDA;       // 1111 1011 1101 1010  BP2＋方向       12パルス
Data2Bp[2].Bp2m = 0;            // 0000 0000 0000 0000  BP2－方向        0パルス

Data2Bp[3].Bp1p = 0;            // 0000 0000 0000 0000  BP1＋方向        0パルス
Data2Bp[3].Bp1m = 0x1CF2;       // 0001 1100 1111 0010  BP1－方向        8パルス
Data2Bp[3].Bp2p = 0xFFFF;       // 1111 1111 1111 1111  BP2＋方向       16パルス
Data2Bp[3].Bp2m = 0;            // 0000 0000 0000 0000  BP2－方向        0パルス
```

例２．定義と初期化を同時に行う場合

```
// ２軸ＢＰ補間データ     BP1P,   BP1M,   BP2P,   BP2M  (MCX314As取説 図2.32のデータ)
DATA_2BP[] Data2Bp = new DATA_2BP[]
{
        new DATA_2BP(0x0000, 0x2BFF, 0xFFD4, 0x0000),
        new DATA_2BP(0xF6FE, 0x0000, 0x000F, 0x3FC0),
        new DATA_2BP(0x1FDB, 0x0000, 0x00FF, 0xFC00),
        new DATA_2BP(0x4000, 0x7FF5, 0x0000, 0x0AFF),
};
```

(4) 補間関数で指定する補間軸(IpAxis)の指定は下記の通りです。
４軸Ｘ、Ｙ、Ｚ、Ｕの中から補間する軸の組み合わせを、次の１６ビットデータの下位６ビットに設定します。２軸補間の場合は第１軸、第２軸に、第３軸補間の場合は第１軸、第２軸、第３軸に設定します。

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0	AX31	AX30	AX21	AX20	AX11	AX10
										第３軸		第２軸		第１軸	

◆各ビットの説明

D1,0　AX11,10　補間ドライブを行う第１軸(主軸)を指定します。軸コードを下表に示します。

軸	コード(２進)
Ｘ	００
Ｙ	０１
Ｚ	１０
Ｕ	１１

第１軸：Ｘ、第２軸：Ｙ、第３軸：Ｚの例

D5	D4	D3	D2	D1	D0
1	0	0	1	0	0

D3,2　AX21,20　補間ドライブを行う第２軸を上表に示すコードで指定します。

D5,4　AX31,30　３軸補間ドライブを行う第３軸を上表に示すコードで指定します。
２軸補間ドライブの時は使用しないので、何をセットしても構いません。

◆Ｃ＃．ＮＥＴの場合、次の既定値を使い補間軸を指定して下さい。

```
public enum IP_AXIS : int
{
        IP_X    =0,     // 補間軸Ｘ
        IP_Y    =1,     // 補間軸Ｙ
        IP_Z    =2,     // 補間軸Ｚ
        IP_U    =3,     // 補間軸Ｕ
}
```

軸	使用例
Ｘ	IP_AXIS.IP_X
Ｙ	IP_AXIS.IP_Y
Ｚ	IP_AXIS.IP_Z
Ｕ	IP_AXIS.IP_U

補間軸の指定方法は、次の例のように、ビット演算を用いて設定します。

（使用例）第１軸：Ｘ、第２軸：Ｙ、第３軸：Ｚの３軸補間の場合、次のようにビット演算をして設定してください。

```
IpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y<<2 | IP_AXIS.IP_Z<<4 ;
```

(5) 連続補間関数実行時の速度変更について

連続補間関数は、補間実行中に速度変更を行う事ができます。各セグメント毎に速度を設定できます。
補間実行中に速度変更を行う場合は、関数のパラメータSpdChgFlgにTRUE(True)を設定します。

◆各セグメント毎の速度の設定方法

DATA_2CIPのSpeed、あるいはDATA_3CIPのSpeed に、各セグメントの速度を設定します。
・前のセグメントと異なる速度にする場合は、1～8000 の速度を設定します。
・前のセグメントと同じ速度のままにする場合は、0 を設定します。

◆速度を変更するタイミングについて

DDATA_2CIPのSpeed、あるいはDATA_3CIPのSpeed設定値が 1～8000 の場合の補間関数の処理内容について説明します。

１つ目のセグメントについては、セグメント実行前に速度を設定します。
２つ目以降のセグメントについては、該当セグメントが開始した直後(その次のセグメントが書き込み可能になった時)に速度を設定します。
例えば、２つ目が開始し３つ目のセグメントが書き込み可能になった時に、２つ目のセグメントの速度を設定します。
この為、２つ目が開始しても２つ目の速度を設定するまでの間は１つ目の速度で実行されますので、ご注意下さい。

9.1.3.3 使用方法

■API関数宣言

API関数宣言は、下記の場所で行っています。

VC・・・・MC8043P_DLL.H
VB・・・・MC8043P_DLL.bas
VB.NET・・MC8043P_DLL.vb
C#.NET・・取扱説明書または入力支援を利用して下さい。

■使用方法

(1) 開始処理・・・各関数を使用する前にOpenMC8043Pを一度実行して下さい。
(2) 終了処理・・・プログラム終了時にCloseMC8043P、又はCloseAllMC8043Pを実行して下さい。

■関数使用時の注意

(1) VC,VB,C#（全ての言語）について
①OpenMC8043P関数実行前に各関数を実行した場合の動作保証はできません。
②接続していないMC8043Pのボード番号を誤って指定した場合も、各関数の動作の保証はできません。

(2) VC,C#のみ
①割り込み処理関数を使用する場合はWindowsの性格上、割り込み発生からユーザー定義ルーチンへ制御が移行するまでの時間を保証することは出来ません。
②割込みを行う場合は、割込みユーザー関数（SetEventMC8043Pで指定した関数）を実行中にクローズ処理（CloseMC8043P又はCloseAllMC8043P）を実行しないで下さい。
クローズ処理を行う場合は、必ず、割込みユーザー関数が終了している状態で行って下さい。

■ＶＣにて割込みを処理する方法（ＶＣのみ）

①SetEventMC8043P関数にて、割込みを処理する関数のアドレスを設定します。その際、引数を１つ指定できます。
複数枚ボードを使用する場合は、下記のように設定します。

（ボード番号０の場合）

```
        SetEventMC8043P(0,(LPTHREAD_START_ROUTINE)MC8043P_EventProc0, NULL);
                                        //  関数のアドレスと引数を設定
        Nmc_WriteReg1(0, AXIS_ALL, 0x8000);    //  停止時割込発生(全軸)
        ・・・
```

（ボード番号１の場合）

```
        SetEventMC8043P(1,(LPTHREAD_START_ROUTINE)MC8043P_EventProc1, lpParameter);
                                        //  関数のアドレスと引数を設定
        Nmc_WriteReg1(1, AXIS_ALL, 0x8000);    //  停止時割込発生(全軸)
        ・・・
```

②割込みを処理する関数では、各ボードの割込み要因を確認します。RR3の割込み要因はReadEventMC8043P関数にて取得します。

（ボード番号０の割込みユーザー関数）

```
void MC8043P_EventProc0(void)
{
        long Rr3X, Rr3Y, Rr3Z, Rr3U;
        ReadEventMC8043P(0, &Rr3X,&Rr3Y,&Rr3Z,&Rr3U);
        ・・・・
}
```

（ボード番号１の割込みユーザー関数）

```
void MC8043P_EventProc1(LPVOID lpParameter)
{
        long Rr3X, Rr3Y, Rr3Z, Rr3U;
        ReadEventMC8043P(1, &Rr3X,&Rr3Y,&Rr3Z,&Rr3U);
              ・・・・
}
```

③割込みを処理する関数の設定を解除する場合は ResetEventMC8043P関数を使用します。
この関数を実行すると、ボードで割込みが発生しても割込みユーザー関数は呼び出されません。

■Ｃ＃にて割込みを処理する方法（Ｃ＃のみ）

①MC8043P.SetEventMC8043Pにて、割込みを処理するメソッドを設定します。引数は指定できません。
　複数枚ボードを使用する場合は、下記のように設定します。

（ボード番号０の場合）

```
        MC8043P.callback[0] = new MC8043P.UserThread(isr);       // isrは割込みユーザー関数
        bool ret = MC8043P.SetEventMC8043P(0, MC8043P.callback[0]); // 第１引数にボード番号0を指定
                                                                 //  関数のアドレスを設定
        MC8043P.Nmc_WriteReg1(0, AXIS.ALL, 0x8000);              //  停止時割込発生(全軸)
        ・・・
```

（ボード番号１の場合）

```
        MC8043P.callback[1] = new MC8043P.UserThread(isr2);      // isrは割込みユーザー関数
        bool ret = MC8043P.SetEventMC8043P(1, MC8043P.callback[1]); // 第１引数にボード番号1を指定
                                                                 //  関数のアドレスを設定
        MC8043P.Nmc_WriteReg1(1, AXIS.ALL, 0x8000);              //  停止時割込発生(全軸)
        ・・・
```

②割込処理をする関数では、各ボードの割込み要因を確認します。RR3の割込み要因はMC8043P.ReadEventMC8043Pにて取得します。

（ボード番号０の割込みユーザー関数）

```
static void isr()
{
        int Rr3X, Rr3Y, Rr3Z, Rr3U;
        MC8043P.ReadEventMC8043P(0, &Rr3X,&Rr3Y,&Rr3Z,&Rr3U);
        ・・・・
}
```

（ボード番号１の割込みユーザー関数）

```
static void isr2()
{
        int Rr3X, Rr3Y, Rr3Z, Rr3U;
        MC8043P.ReadEventMC8043P(1, &Rr3X,&Rr3Y,&Rr3Z,&Rr3U);
              ・・・・
}
```

③割込処理をする関数の設定を解除する場合は、MC8043P. ResetEventMC8043Pメソッドを使用します。
　このメソッドを実行すると、ボードで割込みが発生しても割込みユーザー関数のメソッドは呼び出されません。

■連続補間について

MC8043Pで連続補間を行う場合、MCX314As取扱説明書の「2.4.5 連続補間」などを必ず参照し、その章に記載している処理をアプリケーションで行ってください。また、連続補間関数（注１）ではこれらの一部の処理をDLLで行っていますので、この連続補間関数を使用して連続補間の処理を行うこともできます。但し、連続補間関数を使用する場合はいくつか注意事項がありますので、ご注意ください。

注１：Nmc_2CIPExec, Nmc_3CIPExec, Nmc_2CIPExec_BG, Nmc_3CIPExec_BG

連続補間関数を使用する場合の注意事項：

連続補間関数は、次のセグメントの終点や中心点などを書き込み、補間命令書き込み後、エラーチェック行い、エラー発生の場合は関数を終了します。エラーが発生していない場合は、次のセグメントデータ書き込み可能チェック（RR0のD9ビットチェック）を行い、可能になった場合は次のセグメントデータや補間命令を書き込みます。本関数は、連続補間が終了するまでこの処理を繰り返します。エラーチェックと次のセグメントデータ書き込み可能チェック処理などがDLL内部で常にループしているため、本関数実行中にアプリケーションで他の処理も行いたい場合は、本関数の使用が適さない場合があります。この場合は、連続補間関数は使用せず、MCX314As取扱説明書を参考にしてアプリケーションで連続補間の処理を作成してください。
連続補間の加減速ドライブについてはMCX314As取扱説明書の「2.4.6 加減速ドライブでの補間」を参照してください。
また、連続補間関数を使用する場合は、関数を実行する前に初速度を８０００に設定してください。（本関数実行中に初速度を変更しないでください。）この場合、各セグメント内は定速ドライブになります。

■ＢＰ補間（ビットパターン補間）について

MC8043PでＢＰ補間を行う場合、MCX314As取扱説明書の「2.4.3 ビットパターン補間」などを必ず参照し、その章に記載している処理をアプリケーションで行ってください。また、ＢＰ補間関数（注２）ではこれらの一部の処理をDLLで行っていますので、このＢＰ補間関数を使用してＢＰ補間の処理を行うこともできます。但し、ＢＰ補間関数を使用する場合はいくつか注意事項がありますので、ご注意ください。

注２：Nmc_2BPExec, Nmc_3BPExec, Nmc_2BPExec_BG, Nmc_3BPExec_BG

ＢＰ補間関数を使用する場合の注意事項：

ＢＰ補間関数は、次のＢＰデータを書き込み、補間命令書き込み後、エラーチェック行い、エラー発生の場合は関数を終了します。エラーが発生していない場合は、スタックカウンタが２以下になったかチェック（RR0のD14,13ビットチェック）を行い、２以下になった場合は次のＢＰデータを書き込みます。本関数は、ＢＰ補間が終了するまでこの処理を繰り返します。
エラーチェックとスタックカウンタチェック処理などがDLL内部で常にループしているため、本関数実行中にアプリケーションで他の処理も行いたい場合は、本関数の使用が適さない場合があります。この場合は、ＢＰ補間関数は使用せず、MCX314As取扱説明書を参考にしてアプリケーションでＢＰ補間の処理を作成してください。また、ＢＰ補間の加減速ドライブについてはMCX314As取扱説明書の「2.4.6 加減速ドライブでの補間」を参照してください。

■補間関数使用時の注意

(1) 下記の補間関数について、一度に実行できるのは１つの補間関数のみです。
補間関数実行中に、別の補間関数は実行できません。実行した場合、エラーが返ります。

Nmc_2BPExec	Nmc_2BPExec_BG	Nmc_2CIPExec	Nmc_2CIPExec_BG
Nmc_3BPExec	Nmc_3BPExec_BG	Nmc_3CIPExec	Nmc_3CIPExec_BG

(2) 上記の補間関数実行中は、下記の処理を実行しないで下さい。

①補間命令（30h～3Dh）の実行
②ＷＲ５補間モードレジスタの変更

(3) 下記の補間関数はバックグラウンドで実行する為、補間関数開始時に補間データ用のメモリを確保し、ユーザーが指定した補間データをコピーします。そして、バックグラウンドで実行していた補間処理が終了する時にそのメモリを解放し、ユーザーウィンドウにメッセージを送信します。
この為、下記補間関数がバックグラウンドで実行している最中にクローズ処理（CloseMC8043P 又は CloseAllMC8043P）を実行しないで下さい。
また、下記補間関数がバックグラウンドで実行している最中にアプリケーションを終了しないで下さい。
補間関数の実行を途中で止めたい時は、補間中断関数(Nmc_IPStop)を実行し、必ず中断メッセージを受け取って下さい。

Nmc_2BPExec_BG	Nmc_2CIPExec_BG
Nmc_3BPExec_BG	Nmc_3CIPExec_BG

(4) 補間関数にて補間実行中、速度が速い場合は次のデータセットが間に合わず、補間が停止する場合があります。
比較的安定して補間停止が起こらない条件を測定しました。
補間関数実行中にアプリケーションを切り替えたり、何かイベントが発生する場合と発生しない場合では停止する速度が異なります。また、連続補間は、１回の移動量によって、停止する速度は変わってきます。

測定結果：
次の条件では、下記ドライブ速度の場合、補間が停止しませんでした。
それを超えるドライブ速度の時は、補間が停止した事がありました。

[測定環境]
ＯＳ　：WindowsXP SP1
ＣＰＵ　：Celeron(R) CPU 2.53 GHz

◆ 実行関数　：Nmc_2BPExec

①補間関数実行中にアプリケーションを切り替えた場合

補間データ数　：100
安定したドライブ速度　：70 PPS

②補間関数実行中にウィンドウを全く触らなかった場合

補間データ数　：1,000
安定したドライブ速度　：600 PPS

◆ 実行関数　：Nmc_2CIPExec

①補間関数実行中にアプリケーションを切り替えた場合

１回の移動量　：1,000パルス
補間データ数　：100
安定したドライブ速度　：5,000PPS

②補間関数実行中にウィンドウを全く触らなかった場合

１回の移動量　：1,000パルス
補間データ数　：500
安定したドライブ速度　：30,000 PPS

■マルチスレッドアプリケーション開発時の注意

ここでは、マルチスレッドで動作するアプリケーションを開発する際の注意事項を説明します。

Nmc_xxx の関数では、軸切り替え処理、ＷＲ６，ＷＲ７にデータを書き込む処理、ＲＲ６，ＲＲ７にデータを読み出す処理を実行している関数があります。それぞれの Nmc_xxx 関数は次の通りです。

◆軸切り替え処理を実行している関数

Nmc_Reset	Nmc_Command	Nmc_Command_IP			
Nmc_WriteReg0	Nmc_WriteReg1	Nmc_WriteReg2	Nmc_WriteReg3		
Nmc_ReadReg1	Nmc_ReadReg2				
Nmc_Range	Nmc_Jerk	Nmc_Acc	Nmc_Dec	Nmc_StartSpd	Nmc_Speed
Nmc_Pulse	Nmc_Pulse_VB	Nmc_DecP	Nmc_DecP_VB	Nmc_Center	Nmc_Lp
Nmc_Ep	Nmc_CompP	Nmc_CompM	Nmc_AccOfst	Nmc_DJerk	Nmc_HomeSpd
Nmc_ExpMode	Nmc_SyncMode				
Nmc_ReadLp	Nmc_ReadEp	Nmc_ReadSpeed	Nmc_ReadAccDec	Nmc_ReadSyncBuff	
Nmc_2BPExec	Nmc_3BPExec	Nmc_2BPExec_BG	Nmc_3BPExec_BG		
Nmc_2CIPExec	Nmc_3CIPExec	Nmc_2CIPExec_BG	Nmc_3CIPExec_BG		
Nmc_WriteRegSetAxis	Nmc_ReadRegSetAxis	Nmc_WriteData	Nmc_WriteData2	Nmc_ReadData	

◆ＷＲ６，ＷＲ７にデータを書き込む処理を実行している関数

Nmc_Range	Nmc_Jerk	Nmc_Acc	Nmc_Dec	Nmc_StartSpd	Nmc_Speed
Nmc_Pulse	Nmc_Pulse_VB	Nmc_DecP	Nmc_DecP_VB	Nmc_Center	Nmc_Lp
Nmc_Ep	Nmc_CompP	Nmc_CompM	Nmc_AccOfst	Nmc_DJerk	Nmc_HomeSpd
Nmc_ExpMode	Nmc_SyncMode	Nmc_WriteData	Nmc_WriteData2		
Nmc_2CIPExec	Nmc_3CIPExec	Nmc_2CIPExec_BG	Nmc_3CIPExec_BG		
Nmc_WriteReg6	Nmc_WriteReg7				

◆ＲＲ６，ＲＲ７にデータを読み出す処理を実行している関数

Nmc_ReadLp	Nmc_ReadEp	Nmc_ReadSpeed	Nmc_ReadAccDec	Nmc_ReadSyncBuff	Nmc_ReadData

ＷＲ１～ＷＲ３書き込み、ＲＲ１～ＲＲ２読み出し、データ書き込み命令、データ読み出し命令を実行する場合、基本的には次の Nmc_xxx 関数を使用して下さい。

◆ＷＲ１～ＷＲ３書き込み

Nmc_WriteReg1	Nmc_WriteReg2	Nmc_WriteReg3	Nmc_WriteRegSetAxis

◆ＲＲ１～ＲＲ２読み出し

Nmc_ReadReg1	Nmc_ReadReg2	Nmc_ReadRegSetAxis

◆データ書き込み命令

Nmc_Range	Nmc_Jerk	Nmc_Acc	Nmc_Dec	Nmc_StartSpd	Nmc_Speed
Nmc_Pulse	Nmc_Pulse_VB	Nmc_DecP	Nmc_DecP_VB	Nmc_Center	Nmc_Lp
Nmc_Ep	Nmc_CompP	Nmc_CompM	Nmc_AccOfst	Nmc_DJerk	Nmc_HomeSpd
Nmc_ExpMode	Nmc_SyncMode	Nmc_WriteData	Nmc_WriteData2		

◆データ読み出し命令

Nmc_ReadLp	Nmc_ReadEp	Nmc_ReadSpeed	Nmc_ReadAccDec	Nmc_ReadSyncBuff	Nmc_ReadData

これらの Nmc_xxx 関数を使用せずに同じ処理を行った場合、マルチスレッド環境では注意が必要です。

（１）例えば、ＷＲ１書き込み時には Nmc_WriteReg1 を使用しますが、それ以外の方法としては下記の方法があります。

```
① OutpMC8043P(No, MCX314_WR0,  0x010F); // Ｘ軸に切り替える
② OutpMC8043P(No, MCX314_WR1,  Data);   // ＷＲ１書き込み
```

この場合、①と②の間に、軸を切り替える Nmc_xxx 関数が実行された場合、別の軸のＷＲ１にデータが書かれてしまいます。

（２）例えば、速度設定時には Nmc_Speed を使用しますが、それ以外の方法としては下記の方法があります。

```
① OutpMC8043P( No, MCX314_WR6, Data );  // ＷＲ６書き込み
② OutpMC8043P( No, MCX314_WR0, 0x0105 ); // ＷＲ６データをＸ軸の速度に設定する
```

また、次のような関数でも同じ処理を実行できます。

```
③ Nmc_WriteReg6(No, Data);              // ＷＲ６書き込み
④ Nmc_Command(No, AXIS_X, 0x05);        // ＷＲ６データをＸ軸の速度に設定する
```

この場合、①と②の間、③と④の間に、ＷＲ６，ＷＲ７にデータを書き込む Nmc_xxx 関数が実行された場合、別のデータが速度に書かれてしまいます。

（３）例えば、論理位置カウンタ読み出し時には Nmc_ReadLp を使用しますが、それ以外の方法としては下記の方法があります。

```
① OutpMC8043P( No, MCX314_WR0, 0x0110 ); // Ｘ軸の論理位置カウンタをＲＲ６，ＲＲ７に読み出す
② d6 = InpMC8043P( No, MCX314_RR6 );    // ＲＲ６を読み出す
③ d7 = InpMC8043P( No, MCX314_RR7 );    // ＲＲ７を読み出す
```

この場合、①と②の間、②と③の間に、ＲＲ６，ＲＲ７にデータを読み出す Nmc_xxx 関数が実行された場合、ここでは別のデータが読み出されてしまいます。

このように、マルチスレッド環境では、目的の処理を実行するのに２回以上 API 関数をコールする場合は、そのような処理を行わないようにするか、あるいは、排他制御を行う必要があります。

Nmc_xxx、OutpMC8043P の関数を１回コールして処理が完結する場合は、マルチスレッド環境でも問題なく動作します。
Nmc_xxx、OutpMC8043P の各関数同士は排他制御を行っています。

9.1.4　ＡＰＩ（ MC8041Pドライバで使用していた関数をサポート ）

MC8043P.SYS、MC8043P.DLL がアプリケーションに提供するＡＰＩ。

MC8041Pのアプリケーションを既に開発されたお客様の為に、当ドライバでは、下記のMC8041P関数をサポートしています。
下記関数を使用して、MC8043Pに対してアクセスする事が可能です。

MC8043Pのアプリケーションを新規に開発される方は、9.1.3 API（MC8043Pドライバ関数）の方を使用して下さい。

9.1.4.1　ＶＣ＋＋（ MC8043Pを１枚で使用する場合 ）

VC++でプログラミングし、MC8043Pを１枚で使用する場合は、以下の関数を使用します。
下記関数を使用する場合、ボード上のロータリースイッチの値を０に設定して下さい。

関数名	機能　及び　内容
OpenCard	MC8043Pの使用を開始する。ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。 入力パラメータ：void WINAPI　割り込み処理関数アドレス（割込みを使用しない場合はNULL） 戻り値　：HANDLE　オープンに成功するとドライバのハンドルが戻る、失敗するとNULL 〈使用例〉 status = OpenCard(isr);　// 割り込みを使用し、割り込み関数にisrを指定した場合 status = OpenCard(NULL);　// 割り込みを使用しない場合
CloseCard	MC8043Pの使用を終了する。ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。 入力パラメータ：なし 戻り値　：BOOL　クローズに成功するとTRUE、失敗するとFALSE 〈使用例〉 CloseCard();
OutW	出力ポートに１ワード書き込む。ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。 入力パラメータ：WORD　ライトレジスタ番号（WR0～WR7） int　書き込むデータ 戻り値　：なし 〈使用例〉 OutW(WR0, 0x8000);　// ボードのソフトリセット **注意：** ライトレジスタ番号の指定はヘッダ定義ファイルに宣言されている WR0～WR7 を使用してください。
InW	入力ポートから１ワード読み出す。ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。 入力パラメータ：WORD　リードレジスタ番号（RR0～RR7） 戻り値　：WORD　入力ポートから読み込んだ１ワード 〈使用例〉 data = InW(RR0);　// リードレジスタ RR0 の読み出し **注意：** リードレジスタ番号の指定はヘッダ定義ファイルに宣言されている RR0～RR7 を使用してください。 ＲＲ３レジスタのデータ読み出しに関しては、ReadRR3関数の説明を参考にして下さい。
ReadRR3	MC8043Pが割込みイベントを発生した直後のRR3の値(読出し後クリア)を読み出す。 ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。 入力パラメータ：　WORD＊　Ｘ軸のＲＲ３を格納する変数へのポインタ WORD＊　Ｙ軸のＲＲ３を格納する変数へのポインタ WORD＊　Ｚ軸のＲＲ３を格納する変数へのポインタ WORD＊　Ｕ軸のＲＲ３を格納する変数へのポインタ 戻り値　：なし 〈使用例〉 WORD　Rr3X, Rr3Y, Rr3Z, Rr3U; ReadRR3(&Rr3X, &Rr3Y, &Rr3Z, &Rr3U); **注意：** MC8043Pが割込を発生した直後ドライバ内でＲＲ３を読み出してしまうのでＲＲ３はクリアされてしまいます。 割込み発生直後のＲＲ３を確認する場合はこの関数を使用してください。

9.1.4.2 ＶＣ＋＋（ MC8043Pを複数枚で使用する場合 ）

VC++でプログラミングし、MC8043Pを複数枚で使用する場合は、以下の関数を使用します。

関数名	機能　及び　内容

OpenCard_N	MC8043Pの使用を開始する。
	
	入力パラメータ：int　　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値
	　　void WINAPI　割り込み処理関数アドレス（割込みを使用しない場合はNULL）
	戻り値　　：HANDLE　　オープンに成功するとドライバのハンドルが戻る、失敗するとNULL
	
	〈使用例〉
	　status = OpenCard_N(1, isr);　//ボード番号０に対し割り込みを使用し
	　　//割り込み関数に isr を指定した場合
	　status = OpenCard_N(1, NULL);　//ボード番号０、割り込みを使用しない場合

CloseCard_N	MC8043Pの使用を終了する。 入力パラメータ：int　　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値 戻り値　　：BOOL　　クローズに成功するとTRUE、失敗するとFALSE 〈使用例〉 　CloseCard_N(1);　　//ボード番号０
OutW_N	出力ポートに１ワード書き込む。 入力パラメータ：int　　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値 　　WORD　　ライトレジスタ番号（WR0～WR7） 　　int　　書き込むデータ 戻り値　　：なし 〈使用例〉 　OutW_N(1, WR0, 0x8000);　// ボードのソフトリセット **注意：** 　ライトレジスタ番号の指定はヘッダ定義ファイルに宣言されている WR0～WR7 を使用してください。
InW_N	入力ポートから１ワード読み出す。 入力パラメータ：int　　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値 　　WORD　　リードレジスタ番号（RR0～RR7） 戻り値　　：WORD　　入力ポートから読み込んだ１ワード 〈使用例〉 　data = InW_N(1, RR0);　　// リードレジスタ RR0 の読み出し **注意：** 　リードレジスタ番号の指定はヘッダ定義ファイルに宣言されている RR0～RR7 を使用してください。 　ＲＲ３レジスタのデータ読み出しに関しては、ReadRR3_N関数の説明を参考にして下さい。
CloseCard_all	全てのMC8043Pの使用を終了する。 入力パラメータ：なし 戻り値　　：BOOL　　クローズに成功するとTRUE、失敗するとFALSE 〈使用例〉 　CloseCard_all();
ReadRR3_N	MC8043Pが割込みイベントを発生した直後のRR3の値(読出し後クリア)を読み出す。 入力パラメータ：int　　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値 　　WORD＊　　Ｘ軸のＲＲ３を格納する変数へのポインタ 　　WORD＊　　Ｙ軸のＲＲ３を格納する変数へのポインタ 　　WORD＊　　Ｚ軸のＲＲ３を格納する変数へのポインタ 　　WORD＊　　Ｕ軸のＲＲ３を格納する変数へのポインタ 戻り値　　：なし 〈使用例〉 　　ReadRR3_N(1, &Rr3X, &Rr3Y, &Rr3Z, &Rr3U);　// ボード番号０のRR3の内容をリード **注意：** 　MC8043Pが割込を発生した直後ドライバ内でＲＲ３を読み出してしまうのでＲＲ３はクリアされてしまいます。 　割込み発生直後のＲＲ３を確認する場合はこの関数を使用してください。

9.1.4.3 ＶＢ６．０（ MC8043Pを１枚で使用する場合 ）

VB6.0でプログラミングし、MC8043Pを１枚で使用する場合は、以下の関数を使用します。
下記関数を使用する場合、ボード上のロータリースイッチの値を０に設定して下さい。

関数名	機能　及び　内容
OpenCard	MC8043Pの使用を開始する。ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。 入力パラメータ　：　０＆　　固定 戻り値　　：　As Long　　オープンに成功するとドライバのハンドルが戻る、失敗するとNULL 〈使用例〉 　status = OpenCard(0&)　　'０＆は固定、ボード０をオープン
CloseCard	MC8043Pの使用を終了する。ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。 入力パラメータ：　なし 戻り値　　：　As Long　　クローズに成功すると０以外、失敗すると０ 〈使用例〉 　status = CloseCard()　　'ボード０をクローズ
OutW	出力ポートに１ワード書き込む。ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。 入力パラメータ：　ByVal As Intger　ライトレジスタ番号（WR0～WR7） 　　ByVal As Long　書き込むデータ 戻り値　　：　なし 〈使用例〉 　Call　OutW(WR0, &H8000)　　'ボードのソフトリセット **注意：** 　ライトレジスタ番号の指定は標準モジュールで定義した WR0～WR7 を使用してください。
InW	入力ポートから１ワード読み出す。ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。 入力パラメータ：　ByVal As Integer　リードレジスタ番号（RR0～RR7） 戻り値　　：　As Long　　入力ポートから読み込んだ１ワード 〈使用例〉 　data = InW(RR0)　　'リードレジスタ RR0 の読み出し **注意：** 　リードレジスタ番号の指定は標準モジュールで定義した RR0～RR7 を使用してください。

9.1.4.4 ＶＢ６．０（ MC8043Pを複数枚で使用する場合 ）

VB6.0でプログラミングし、MC8043Pを複数枚で使用する場合は、以下の関数を使用します。

関数名	機能　及び　内容
OpenCard_N	MC8043Pの使用を開始する。 入力パラメータ：ByVal As Long　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値 　　０＆　固定 戻り値　　：As Long　オープンに成功するとドライバのハンドルが戻る、失敗するとNULL 〈使用例〉 　status = OpenCard_N(1, 0&)　'第１引数が１の時、ボード番号０オープン，第２引数は０＆固定
CloseCard_N	MC8043Pの使用を終了する。 入力パラメータ：ByVal As Long　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値 戻り値　　：As Long　クローズに成功すると０以外、失敗すると０ 〈使用例〉 　status = CloseCard_N(1)　'ボード番号０クローズ
OutW_N	出力ポートに１ワード書き込む。 入力パラメータ：ByVal As Long　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値 　　ByVal As Intger　ライトレジスタ番号（WR0～WR7） 　　ByVal As Long　書き込むデータ 戻り値　　：なし 〈使用例〉 　Call　OutW_N(1, WR0, &H8000)　'ボードのソフトリセット **注意：** 　ライトレジスタ番号の指定は標準モジュールで定義した WR0～WR7 を使用してください。
InW_N	入力ポートから１ワード読み出す。 入力パラメータ：ByVal As Long　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値 　　ByVal As Integer　リードレジスタ番号（RR0～RR7） 戻り値　　：As Long　入力ポートから読み込んだ１ワード 〈使用例〉 　data = InW_N(1, RR0)　'リードレジスタ RR0 の読み出し **注意：** 　リードレジスタ番号の指定は標準モジュールで定義した RR0～RR7 を使用してください。
CloseCard_all	全てのMC8043Pの使用を終了する。 入力パラメータ：なし 戻り値　　：As Long　クローズに成功すると０以外、失敗すると０ 〈使用例〉 　status = CloseCard_all( )

9.1.4.5 ＶＢ.ＮＥＴ２００３（ MC8043Pを１枚で使用する場合 ）

VB.NET 2003でプログラミングし、MC8043Pを１枚で使用する場合は、以下の関数を使用します。
下記関数を使用する場合、ボード上のロータリースイッチの値を０に設定して下さい。

関数名	機能　及び　内容
OpenCard	MC8043Pの使用を開始する。ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。 入力パラメータ　：　０　　固定 戻り値　　：　As Integer　　オープンに成功するとドライバのハンドルが戻る、失敗するとNULL 〈使用例〉 　status = OpenCard(0)　　'０は固定、ボード０をオープン
CloseCard	MC8043Pの使用を終了する。ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。 入力パラメータ：　なし 戻り値　　：　As Integer　　クローズに成功すると０以外、失敗すると０ 〈使用例〉 　status = CloseCard()　　'ボード０をクローズ
OutW	出力ポートに１ワード書き込む。ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。 入力パラメータ：　ByVal As Short　ライトレジスタ番号（WR0～WR7） 　　ByVal As Integer　書き込むデータ 戻り値　　：　なし 〈使用例〉 　Call　OutW(WR0, &H8000)　　'ボードのソフトリセット **注意：** 　ライトレジスタ番号の指定は標準モジュールで定義した WR0～WR7 を使用してください。
InW	入力ポートから１ワード読み出す。ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。 入力パラメータ：　ByVal As Short　リードレジスタ番号（RR0～RR7） 戻り値　　：　As Integer　　入力ポートから読み込んだ１ワード 〈使用例〉 　data = InW(RR0)　　'リードレジスタ RR0 の読み出し **注意：** 　リードレジスタ番号の指定は標準モジュールで定義した RR0～RR7 を使用してください。

9.1.4.6 ＶＢ.ＮＥＴ２００３（ MC8043Pを複数枚で使用する場合 ）

VB.NET 2003でプログラミングし、MC8043Pを複数枚で使用する場合は、以下の関数を使用します。

関数名	機能　及び　内容
OpenCard_N	MC8043Pの使用を開始する。 入力パラメータ：ByVal As Integer　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値 　　0　固定 戻り値　　：As Integer　オープンに成功するとドライバのハンドルが戻る、失敗するとNULL 〈使用例〉 　status = OpenCard_N(1, 0)　'第１引数が１の時、ボード番号０オープン，第２引数は０固定
CloseCard_N	MC8043Pの使用を終了する。 入力パラメータ：ByVal As Integer　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値 戻り値　　：As Integer　クローズに成功すると０以外、失敗すると０ 〈使用例〉 　status = CloseCard_N(1)　'ボード番号０クローズ
OutW_N	出力ポートに１ワード書き込む。 入力パラメータ：ByVal As Integer　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値 　　ByVal As Short　ライトレジスタ番号（WR0～WR7） 　　ByVal As Integer　書き込むデータ 戻り値　　：なし 〈使用例〉 　Call　OutW_N(1, WR0, &H8000)　'ボードのソフトリセット **注意：** 　ライトレジスタ番号の指定は標準モジュールで定義した WR0～WR7 を使用してください。
InW_N	入力ポートから１ワード読み出す。 入力パラメータ：ByVal As Integer　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値 　　ByVal As Short　リードレジスタ番号（RR0～RR7） 戻り値　　：As Integer　入力ポートから読み込んだ１ワード 〈使用例〉 　data = InW_N(1, RR0)　'リードレジスタ RR0 の読み出し **注意：** 　リードレジスタ番号の指定は標準モジュールで定義した RR0～RR7 を使用してください。
CloseCard_all	全てのMC8043Pの使用を終了する。 入力パラメータ：なし 戻り値　　：As Integer　クローズに成功すると０以外、失敗すると０ 〈使用例〉 　status = CloseCard_all( )

9.1.4.7　C＃（ MC8043Pを１枚で使用する場合 ）

C#でプログラミングし、MC8043Pを１枚で使用する場合は、以下の関数を使用します。
下記関数を使用する場合、ボード上のロータリースイッチの値を０に設定して下さい。

関数名	機能　及び　内容

OpenCard	MC8043Pの使用を開始する。ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。
	
	入力パラメータ：UserThread Callback　割り込み処理関数アドレス
	（割込みを使用しない場合はCallbackにnullを設定）
	戻り値　：System.IntPtr　オープンに成功するとNULL以外が戻る、失敗するとNULL
	
	〈割り込みを使用し、割り込み関数にisrを指定した場合の例〉
	MC8043P.callback[0] = new MC8043P.UserThread(isr);
	status = MC8043P.OpenCard(MC8043P.callback[0]);
	
	〈割込みを使用しない場合の例〉
	MC8043P.callback[0] = null;
	status = MC8043P.OpenCard(MC8043P.callback[0]);

CloseCard	MC8043Pの使用を終了する。ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。 入力パラメータ：なし 戻り値　：bool　クローズに成功するとtrue、失敗するとfalse 〈使用例〉 MC8043P.CloseCard();
OutW	出力ポートに１ワード書き込む。ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。 入力パラメータ：REG_MC8041P　ライトレジスタ番号（WR0～WR7） int　書き込むデータ 戻り値　：なし 〈使用例〉 MC8043P.OutW(REG_MC8041P.WR0, 0x8000);　// ボードのソフトリセット **注意：** ライトレジスタ番号の指定は補足説明(1)参照。
InW	入力ポートから１ワード読み出す。ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。 入力パラメータ：REG_MC8041　リードレジスタ番号（RR0～RR7） 戻り値　：ushort　入力ポートから読み込んだ１ワード 〈使用例〉 data = MC8043P.InW(REG_MC8041.PRR0);　// リードレジスタ RR0 の読み出し **注意：** リードレジスタ番号の指定は補足説明(1)参照。 ＲＲ３レジスタのデータ読み出しに関しては、ReadRR3関数の説明を参考にして下さい。
ReadRR3	MC8043Pが割込みイベントを発生した直後のRR3の値(読出し後クリア)を読み出す。 ボード番号（ボード上のロータリースイッチの値）が０のMC8043Pに対して行う。 入力パラメータ：　ushort＊　Ｘ軸のＲＲ３を格納する変数へのポインタ ushort＊　Ｙ軸のＲＲ３を格納する変数へのポインタ ushort＊　Ｚ軸のＲＲ３を格納する変数へのポインタ ushort＊　Ｕ軸のＲＲ３を格納する変数へのポインタ 戻り値　：なし 〈使用例〉 ushort　Rr3X, Rr3Y, Rr3Z, Rr3U; MC8043P.ReadRR3(&Rr3X, &Rr3Y, &Rr3Z, &Rr3U); **注意：** MC8043Pが割込を発生した直後ドライバ内でＲＲ３を読み出してしまうのでＲＲ３はクリアされてしまいます。 割込み発生直後のＲＲ３を確認する場合はこの関数を使用してください。

9.1.4.8　C＃（ MC8043Pを複数枚で使用する場合 ）

C#でプログラミングし、MC8043Pを複数枚で使用する場合は、以下の関数を使用します。

関数名	機能　及び　内容
OpenCard_N	MC8043Pの使用を開始する。 入力パラメータ：int　　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値 　　UserThread Callback　割り込み処理関数アドレス 　　(割込みを使用しない場合はCallbackにnullを設定) 戻り値　　：System.IntPtr　　オープンに成功するとNULL以外が戻る、失敗するとNULL <ボード番号1で割り込みを使用し、割り込み関数にisrを指定した場合の例> MC8043P.callback[1] = new MC8043P.UserThread(isr); status = MC8043P.OpenCard_N(2,　MC8043P.callback[1]); <ボード番号1で割込みを使用しない場合の例> MC8043P.callback[1] = null; status = MC8043P.OpenCard_N(2, MC8043P.callback[1]);
CloseCard_N	MC8043Pの使用を終了する。 入力パラメータ：int　　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値 戻り値　　：bool　　クローズに成功するとtrue、失敗するとfalse <ボード番号1の使用例> MC8043P.CloseCard_N(2);
OutW_N	出力ポートに１ワード書き込む。 入力パラメータ：int　　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値 　　REG_MC8041P　ライトレジスタ番号（WR0～WR7） 　　int　　書き込むデータ 戻り値　　：なし <ボード番号1の使用例> MC8043P.OutW_N(2, REG_MC8041P.WR0, 0x8000);　// ボードのソフトリセット **注意：** ライトレジスタ番号の指定は補足説明(1)参照。
InW_N	入力ポートから１ワード読み出す。 入力パラメータ：int　　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値 　　REG_MC8041P　リードレジスタ番号（RR0～RR7） 戻り値　　：ushort　　入力ポートから読み込んだ１ワード <ボード番号1の使用例> data = MC8043P.InW_N(2, REG_MC8041P.RR0);　　// リードレジスタ RR0 の読み出し **注意：** リードレジスタ番号の指定は補足説明(1)参照。 ＲＲ３レジスタのデータ読み出しに関しては、ReadRR3_N関数の説明を参考にして下さい。
CloseCard_all	全てのMC8043Pの使用を終了する。 入力パラメータ：なし 戻り値　　：bool　　クローズに成功するとtrue、失敗するとfalse <使用例> MC8043P.CloseCard_all();
ReadRR3_N	MC8043Pが割込みイベントを発生した直後のRR3の値(読出し後クリア)を読み出す。 入力パラメータ：int　　ボード番号（ボード上のロータリースイッチの値(0～9)）＋１の値 　　ushort＊　Ｘ軸のＲＲ３を格納する変数へのポインタ 　　ushort＊　Ｙ軸のＲＲ３を格納する変数へのポインタ 　　ushort＊　Ｚ軸のＲＲ３を格納する変数へのポインタ 　　ushort＊　Ｕ軸のＲＲ３を格納する変数へのポインタ 戻り値　　：なし <ボード番号1の使用例> ushort　Rr3X, Rr3Y, Rr3Z, Rr3U; MC8043P.ReadRR3_N(1, &Rr3X, &Rr3Y, &Rr3Z, &Rr3U);　// ボード番号０のRR3の内容をリード **注意：** MC8043Pが割込を発生した直後ドライバ内でＲＲ３を読み出してしまうのでＲＲ３はクリアされてしまいます。 割込み発生直後のＲＲ３を確認する場合はこの関数を使用してください。

9.1.4.9 注意点

・VC++、VB、C# のどれを使用する場合でも、各関数を使用する前にOpenCard()又はOpenCard_N()を必ず実行してください。
・プログラム終了時にCloseCard(),又はCloseCard_N(),又はCloseCard_all()を実行してください。
・OpenCard()又はOpenCard_N()関数実行前に各関数を実行した場合の動作保証はできません。
・接続していないMC8043Pのボード番号を誤って指定した場合も、各関数の動作の保証はできません。
・VC++、C#のみで割り込みをサポートしていますが、割り込み処理関数を使用する場合はWindowsの性格上、割り込み発生からユーザー定義ルーチンへ制御が移行するまでの時間を保証することは出来ません。
・VC++、C#で割込みを行う場合は、割込みユーザー関数（OpenCard()又はOpenCard_N()で指定した関数）を実行中にクローズ処理（CloseCard(),又はCloseCard_N(),又はCloseCard_all()）を実行しないで下さい。クローズ処理を行う場合は、必ず、割込みユーザー関数が終了している状態で行って下さい。

9.2　ソフトウェアの内容

ソフトウェアのフォルダ構成とファイル内容を以下に示します。

注意：ＣＤ－ＲＯＭからハードディスクにコピーする場合、ファイルやフォルダが読み取り専用になる場合がありますので、必要であれば読み取り専用を解除してから使用して下さい。

```
¥
+---Driver
|        +---MC8043P.sys                デバイスドライバ本体
|        +---MC8043P.inf                デバイスドライバのインストール用プログラム
|        +---MC8043P.dll                デバイスドライバを使用するためのダイナミックリンクライブラリ
|        +---Version.txt                デバイスドライバのバージョン説明ファイル
+---LIB
|    +---VB6
|    |   +---MC8043P_DLL.bas            VB6.0用 MC8043P.DLL Declare宣言、各定義ファイル
|    +---VB.NET2003
|    |   +---MC8043P_DLL.vb             VB.NET2003用 MC8043P.DLL Declare宣言、各定義ファイル
|    +---VC6
|    |   +---MC8043P.lib                VC6.0用 MC8043P.DLLのライブラリファイル
|    |   +---MC8043P_DLL.h              VC6.0用 MC8043P.DLLのヘッダファイル（関数宣言、各定義）
|    |
|    +---Csharp
|        +---Mc8043pWrap.dll            MC8043P Ｃ＃クラスライブラリファイル
|
+---Tool
|    +---Tool8043.exe                   MC8043P評価ツール
|    +---操作説明書.txt                 操作説明書
|    +---ParameterSample                パラメータサンプルファイル
|        +---対称Ｓ字加減速-1.msd       対称Ｓ字加減速パラメータファイル１
|        +---対称Ｓ字加減速-2.msd       対称Ｓ字加減速パラメータファイル２
|        +---対称台形加減速-1.msd       対称台形加減速パラメータファイル１
|        +---対称台形加減速-2.msd       対称台形加減速パラメータファイル２
|        +---非対称Ｓ字加減速-1.msd     非対称Ｓ字加減速パラメータファイル１
|        +---非対称Ｓ字加減速-2.msd     非対称Ｓ字加減速パラメータファイル２
|        +---非対称台形加減速-1.msd     非対称台形加減速パラメータファイル１
|        +---非対称台形加減速-2.msd     非対称台形加減速パラメータファイル２
|        +---readme.txt                 説明書
|
+---Sample
    +---VB6
    |   +---NormallyClose
    |   |   +---Sample A
    |   |       +---FormA.frm           サンプルプログラムＡ（Normally Close）
    |   |       +---MC8043P_DLL.bas     VB6用MC8043P.DLL Declare宣言、各定義ファイル
    |   |       +---Module1.bas         MC8043P制御関数サンプル
    |   |       +---VBSample.vbp        VBサンプルプログラム用プロジェクトファイル（VB6.0）
    |   |       +---exe
    |   |           +---VBSampleA.exe   実行ファイル
    |   |
    |   +---Sample A
    |   |   +---FormA.frm               サンプルプログラムＡ
    |   |   +---MC8043P_DLL.bas         VB6用MC8043P.DLL Declare宣言、各定義ファイル
    |   |   +---Module1.bas             MC8043P制御関数サンプル
    |   |   +---VBSample.vbp            VBサンプルプログラム用プロジェクトファイル（VB6.0）
    |   |   +---exe
    |   |       +---VBSampleA.exe       実行ファイル
    |   |
    |   +---Sample C
    |   |   +---FormC.frm               サンプルプログラムＣ
    |   |   +---MC8043P_DLL.bas         VB6用MC8043P.DLL Declare宣言、各定義ファイル
    |   |   +---VBSample.vbp            VBサンプルプログラム用プロジェクトファイル（VB6.0）
    |   |   +---exe
    |   |       +---VBSampleC.exe       実行ファイル
    |   |
```

```
|   +---Sample E
|   |   +---Form1.frm               サンプルプログラムＥ
|   |   +---MC8043P_DLL.bas         VB6用MC8043P.DLL Declare宣言、各定義ファイル
|   |   +---Module1.bas             MC8043P制御関数サンプル
|   |   +---MC_Sample.vbp           VBサンプルプログラム用プロジェクトファイル (VB6.0)
|   |   +---exe
|   |       +---VBSampleE.exe       実行ファイル
|   |
|   +---Sample G
|   |   +---Form1.frm               サンプルプログラムＧ
|   |   +---MC8043P_DLL.bas         VB6用MC8043P.DLL Declare宣言、各定義ファイル
|   |   +---Module1.bas             MC8043P制御関数サンプル
|   |   +---MC_Sample.vbp           VBサンプルプログラム用プロジェクトファイル (VB6.0)
|   |   +---exe
|   |       +---VBSampleG.exe       実行ファイル
|   |
|   +---Sample H
|   |   +---SampleH.frm             サンプルプログラムＨ
|   |   +---MC8043P_DLL.bas         VB6用MC8043P.DLL Declare宣言、各定義ファイル
|   |   +---Module1.bas             MC8043P制御関数サンプル
|   |   +---SampleH.vbp             VBサンプルプログラム用プロジェクトファイル (VB6.0)
|   |   +---exe
|   |       +---VBSampleH.exe       実行ファイル
|   |
|   +---Sample I
|       +---SampleI.frm             サンプルプログラムＩ
|       +---MC8043P_DLL.bas         VB6用MC8043P.DLL Declare宣言、各定義ファイル
|       +---Module1.bas             MC8043P制御関数サンプル
|       +---SampleI.vbp             VBサンプルプログラム用プロジェクトファイル (VB6.0)
|       +---exe
|           +---VBSampleI.exe       実行ファイル
|
+---VB.NET2003
|   +---NormallyClose
|   |   +---Sample A
|   |       +---FormA.vb            サンプルプログラムＡ (Normally Close)
|   |       +---MC8043P_DLL.vb      VB.NET2003用 MC8043P.DLL Declare宣言、各定義ファイル
|   |       +---Module1.vb          MC8043P制御関数サンプル
|   |       +---VBSample.sln        VBサンプルプログラム用ソリューションファイル (VB.NET2003)
|   |       +---exe
|   |           +---VBSampleA.exe   実行ファイル
|   |
|   +---Sample A
|   |   +---FormA.vb                サンプルプログラムＡ
|   |   +---MC8043P_DLL.vb          VB.NET2003用 MC8043P.DLL Declare宣言、各定義ファイル
|   |   +---Module1.vb              MC8043P制御関数サンプル
|   |   +---VBSample.sln            VBサンプルプログラム用ソリューションファイル (VB.NET2003)
|   |   +---exe
|   |       +---VBSampleA.exe       実行ファイル
|   |
|   +---Sample C
|   |   +---FormC.vb                サンプルプログラムＣ
|   |   +---MC8043P_DLL.vb          VB.NET2003用 MC8043P.DLL Declare宣言、各定義ファイル
|   |   +---VBSample.sln            VBサンプルプログラム用ソリューションファイル (VB.NET2003)
|   |   +---exe
|   |       +---VBSampleC.exe       実行ファイル
|   |
|   +---Sample E
|   |   +---Form1.vb                サンプルプログラムＥ
|   |   +---MC8043P_DLL.vb          VB.NET2003用 MC8043P.DLL Declare宣言、各定義ファイル
|   |   +---Module1.vb              MC8043P制御関数サンプル
|   |   +---MC_Sample.sln           VBサンプルプログラム用ソリューションファイル (VB.NET2003)
|   |   +---exe
|   |       +---VBSampleE.exe       実行ファイル
|   |
```

```
|   +---Sample G
|   |   +---Form1.vb                サンプルプログラムＧ
|   |   +---MC8043P_DLL.vb          VB.NET2003用 MC8043P.DLL Declare宣言、各定義ファイル
|   |   +---Module1.vb              MC8043P制御関数サンプル
|   |   +---MC_Sample.sln           VBサンプルプログラム用ソリューションファイル (VB.NET2003)
|   |   +---exe
|   |       +---VBSampleG.exe       実行ファイル
|   |
|   +---Sample H
|   |   +---SampleH.vb              サンプルプログラムＨ
|   |   +---MC8043P_DLL.vb          VB.NET2003用 MC8043P.DLL Declare宣言、各定義ファイル
|   |   +---Module1.vb              MC8043P制御関数サンプル
|   |   +---SampleH.sln             VBサンプルプログラム用ソリューションファイル (VB.NET2003)
|   |   +---exe
|   |       +---VBSampleH.exe       実行ファイル
|   |
|   +---Sample I
|       +---SampleI.vb              サンプルプログラムＩ
|       +---MC8043P_DLL.vb          VB.NET2003用 MC8043P.DLL Declare宣言、各定義ファイル
|       +---Module1.vb              MC8043P制御関数サンプル
|       +---SampleI.sln             VBサンプルプログラム用ソリューションファイル (VB.NET2003)
|       +---exe
|           +---VBSampleI.exe       実行ファイル
|
+---VC6
|   +---NormallyClose
|   |   +---Sample A
|   |       +---SmapleA.cpp         サンプルプログラムＡ (Normally Close)
|   |       +---MC8043P.LIB         MC8043P.DLLのライブラリファイル
|   |       +---MC8043P_DLL.H       MC8043P.DLLのヘッダファイル (関数宣言、各定義)
|   |       +---VCSample.dsw        VCサンプルプログラム用プロジェクトワークスペース (VC6.0のみ)
|   |       +---exe
|   |           +---VCSampleA.exe   サンプルＡ実行ファイル
|   |
|   +---Sample A
|   |   +---SmapleA.cpp             サンプルプログラムＡ
|   |   +---MC8043P.LIB             MC8043P.DLLのライブラリファイル
|   |   +---MC8043P_DLL.H           MC8043P.DLLのヘッダファイル (関数宣言、各定義)
|   |   +---VCSample.dsw            VCサンプルプログラム用プロジェクトワークスペース (VC6.0のみ)
|   |   +---exe
|   |       +---VCSampleA.exe       サンプルＡ実行ファイル
|   |
|   +---Sample B
|   |   +---SmapleB.cpp             サンプルプログラムＢ
|   |   +---MC8043P.LIB             MC8043P.DLLのライブラリファイル
|   |   +---MC8043P_DLL.H           MC8043P.DLLのヘッダファイル (関数宣言、各定義)
|   |   +---VCSample.dsw            VCサンプルプログラム用プロジェクトワークスペース (VC6.0のみ)
|   |   +---exe
|   |       +---VCSampleB.exe       サンプルＢ実行ファイル
|   |
|   +---Sample C
|   |   +---SmapleC.cpp             サンプルプログラムＣ
|   |   +---MC8043P.LIB             MC8043P.DLLのライブラリファイル
|   |   +---MC8043P_DLL.H           MC8043P.DLLのヘッダファイル (関数宣言、各定義)
|   |   +---VCSample.dsw            VCサンプルプログラム用プロジェクトワークスペース (VC6.0のみ)
|   |   +---exe
|   |       +---VCSampleC.exe       サンプルＣ実行ファイル
|   |
|   +---Sample D
|   |   +---SmapleD.cpp             サンプルプログラムＤ
|   |   +---MC8043P.LIB             MC8043P.DLLのライブラリファイル
|   |   +---MC8043P_DLL.H           MC8043P.DLLのヘッダファイル (関数宣言、各定義)
|   |   +---VCSample.dsw            VCサンプルプログラム用プロジェクトワークスペース (VC6.0のみ)
|   |   +---exe
|   |       +---VCSampleD.exe       サンプルＤ実行ファイル
|   |
```

```
|   +---Sample E                    サンプルプログラムＥ
|   |   +---MC_SAMPLE.cpp           アプリケーションクラスメンバ関数
|   |   +---MC_SAMPLEDlg.cpp        ダイアログクラスメンバ関数
|   |   +---MC_SAMPLE.H             アプリケーションクラス宣言
|   |   +---MC_SAMPLEDlg.H          ダイアログクラス宣言
|   |   +---MC8043P.cpp             MC8043P制御関数サンプル
|   |   +---MC8043P.H               MC8043P制御関数宣言
|   |   +---MC8043P.LIB             MC8043P.DLLのライブラリファイル
|   |   +---MC8043P_DLL.H           MC8043P.DLLのヘッダファイル（関数宣言、各定義）
|   |   +---MC_SAMPLE.dsw           VCサンプルプログラム用プロジェクトワークスペース（VC6.0のみ）
|   |   +---exe
|   |       +---VCSampleE.exe       実行ファイル
|   |
|   +---Sample F                    サンプルプログラムＦ
|   |   +---MC_Sample2.cpp          アプリケーションクラスメンバ関数
|   |   +---MC_Sample2Dlg.cpp       ダイアログクラスメンバ関数
|   |   +---MC_Sample2.H            アプリケーションクラス宣言
|   |   +---MC_Sample2Dlg.H         ダイアログクラス宣言
|   |   +---MC8043P.cpp             MC8043P制御関数サンプル
|   |   +---MC8043P.H               MC8043P制御関数宣言
|   |   +---MC8043P.LIB             MC8043P.DLLのライブラリファイル
|   |   +---MC8043P_DLL.H           MC8043P.DLLのヘッダファイル（関数宣言、各定義）
|   |   +---MC_Sample2.dsw          VCサンプルプログラム用プロジェクトワークスペース（VC6.0のみ）
|   |   +---Sample.bmp              このアプリケーションで実行する連続補間の軌跡図
|   |   +---exe
|   |       +---VCSampleF.exe       実行ファイル
|   |
|   +---Sample G                    サンプルプログラムＧ
|   |   +---MC_SAMPLE.cpp           アプリケーションクラスメンバ関数
|   |   +---MC_SAMPLEDlg.cpp        ダイアログクラスメンバ関数
|   |   +---MC_SAMPLE.H             アプリケーションクラス宣言
|   |   +---MC_SAMPLEDlg.H          ダイアログクラス宣言
|   |   +---MC8043P.LIB             MC8043P.DLLのライブラリファイル
|   |   +---MC8043P_DLL.H           MC8043P.DLLのヘッダファイル（関数宣言、各定義）
|   |   +---MC_SAMPLE.dsw           VCサンプルプログラム用プロジェクトワークスペース（VC6.0のみ）
|   |   +---exe
|   |       +---VCSampleG.exe       実行ファイル
|   |
|   +---Sample H                    サンプルプログラムＨ
|   |   +---SampleH.cpp             アプリケーションクラスメンバ関数
|   |   +---SampleHDlg.cpp          ダイアログクラスメンバ関数
|   |   +---SampleH.H               アプリケーションクラス宣言
|   |   +---SampleHDlg.H            ダイアログクラス宣言
|   |   +---MC8043P.LIB             MC8043P.DLLのライブラリファイル
|   |   +---MC8043P_DLL.H           MC8043P.DLLのヘッダファイル（関数宣言、各定義）
|   |   +---SampleH.dsw             VCサンプルプログラム用プロジェクトワークスペース（VC6.0のみ）
|   |   +---exe
|   |       +---VCSampleH.exe       実行ファイル
|   |
|   +---Sample I                    サンプルプログラムＩ
|       +---SampleI.cpp             アプリケーションクラスメンバ関数
|       +---SampleIDlg.cpp          ダイアログクラスメンバ関数
|       +---SampleI.H               アプリケーションクラス宣言
|       +---SampleIDlg.H            ダイアログクラス宣言
|       +---MC8043P.LIB             MC8043P.DLLのライブラリファイル
|       +---MC8043P_DLL.H           MC8043P.DLLのヘッダファイル（関数宣言、各定義）
|       +---SampleI.dsw             VCサンプルプログラム用プロジェクトワークスペース（VC6.0のみ）
|       +---exe
|           +---VCSampleI.exe       実行ファイル
|
+---Csharp
    +---Sample A
    |   +---Form1.cs                サンプルプログラムＡ　Ｃ＃ソースファイル
    |   +---SampleA.csproj          サンプルプログラムＡプロジェクトワークスペース（C#のみ）
    |   +---exe
    |       +---SampleA.exe         サンプルＡ実行ファイル
    |
```

```
+---Sample B
|   +---Form1.cs                サンプルプログラムＢ Ｃ＃ソースファイル
|   +---SampleB.csproj          サンプルプログラムＢプロジェクトワークスペース (C#のみ)
|   +---exe
|       +---SampleB.exe         サンプルＢ実行ファイル
|
+---Sample C
|   +---Form1.cs                サンプルプログラムＣ Ｃ＃ソースファイル
|   +---SampleC.csproj          サンプルプログラムＣプロジェクトワークスペース (C#のみ)
|   +---exe
|       +---SampleC.exe         サンプルＣ実行ファイル
|
+---Sample D
|   +---Form1.cs                サンプルプログラムＤ Ｃ＃ソースファイル
|   +---SampleD.csproj          サンプルプログラムＤプロジェクトワークスペース (C#のみ)
|   +---exe
|       +---SampleD.exe         サンプルＤ実行ファイル
|
+---Sample E                    サンプルプログラムＥ
|   +---Form1.cs                サンプルプログラムＥ Ｃ＃ソースファイル
|   +---SampleE.csproj          サンプルプログラムＥプロジェクトワークスペース (C#のみ)
|   +---exe
|       +---SampleE.exe         サンプルＥ実行ファイル
|
+---Sample F                    サンプルプログラムＦ
|   +---Form1.cs                サンプルプログラムＦ Ｃ＃ソースファイル
|   +---SampleF.csproj          サンプルプログラムＦプロジェクトワークスペース (C#のみ)
|   +---exe
|       +---SampleF.exe         サンプルＦ実行ファイル
|
+---Sample G                    サンプルプログラムＧ
|   +---Form1.cs                サンプルプログラムＧ Ｃ＃ソースファイル
|   +---SampleG.csproj          サンプルプログラムＧプロジェクトワークスペース (C#のみ)
|   +---exe
|       +---SampleG.exe         サンプルＧ実行ファイル
|
+---Sample H                    サンプルプログラムＨ
|   +---Form1.cs                サンプルプログラムＨ Ｃ＃ソースファイル
|   +---SampleH.csproj          サンプルプログラムＨプロジェクトワークスペース (C#のみ)
|   +---exe
|       +---SampleH.exe         サンプルＨ実行ファイル
|
+---Sample I                    サンプルプログラムＩ
    +---Form1.cs                サンプルプログラムＩ Ｃ＃ソースファイル
    +---SampleI.csproj          サンプルプログラムＩプロジェクトワークスペース (C#のみ)
    +---exe
        +---SampleI.exe         サンプルＩ実行ファイル
```

9.3　開発手順

9.3.1　VC++でアプリケーションを開発する場合(VC++6.0, VC++.NET2003)

MC8043Pアプリケーションでは MC8043P.lib と MC8043P_DLL.h ファイルを使用します。この２ファイルは VC++ 6.0 以降対応です。

■新規にMC8043Pアプリケーションを作成する場合

① ¥Lib¥VC6フォルダに入っている２つのファイル MC8043P.lib と MC8043P_DLL.h を開発するアプリケーションのフォルダにコピーしてください。

② VC++の総合開発環境にて MC8043P_DLL.h をご使用のプロジェクトに追加登録してください。
また、API 関数を使用するソースファイルに MC8043P_DLL.h を include して下さい。

③ VC++6.0 の場合は、[プロジェクト]－[設定]で「リンク」タブを選択し「オブジェクト／ライブラリモジュール」に MC8043P.lib を追加して下さい。（「図9.3-1 VC++6.0 プロジェクトの設定」を参照。）

VC++.NET2003 の場合は、[プロジェクト]－[プロパティ]画面で[リンカ]－[入力]を選択し、「追加の依存ファイル」に MC8043P.lib を追加して下さい。（「図9.3-2 VC++.NET2003 プロジェクトのプロパティ」を参照。）

④ 9.1.3 ＡＰＩ（ MC8043Pドライバ関数 ）の関数を使用してプログラミングを行って下さい。

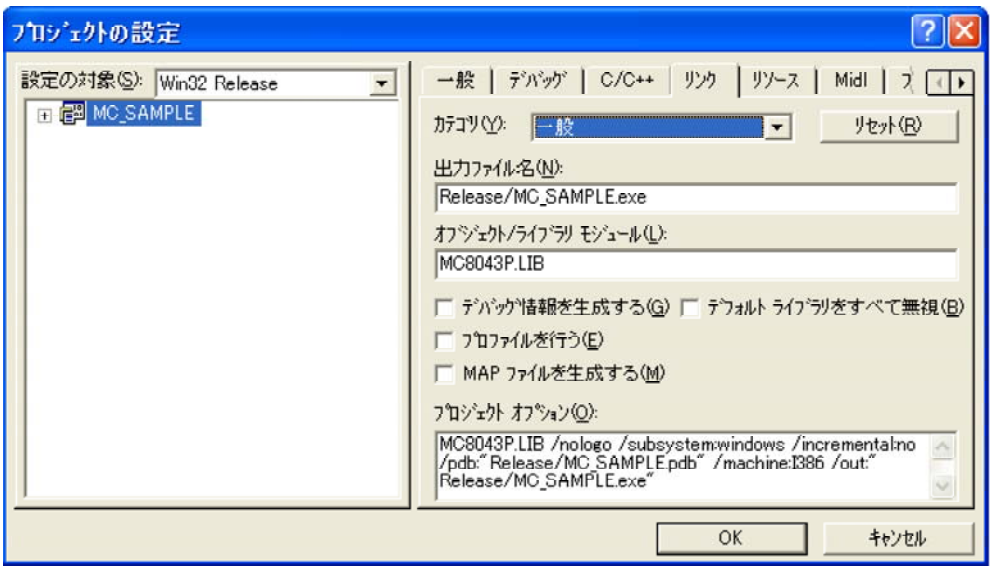


図9.3-1 VC++6.0 プロジェクトの設定

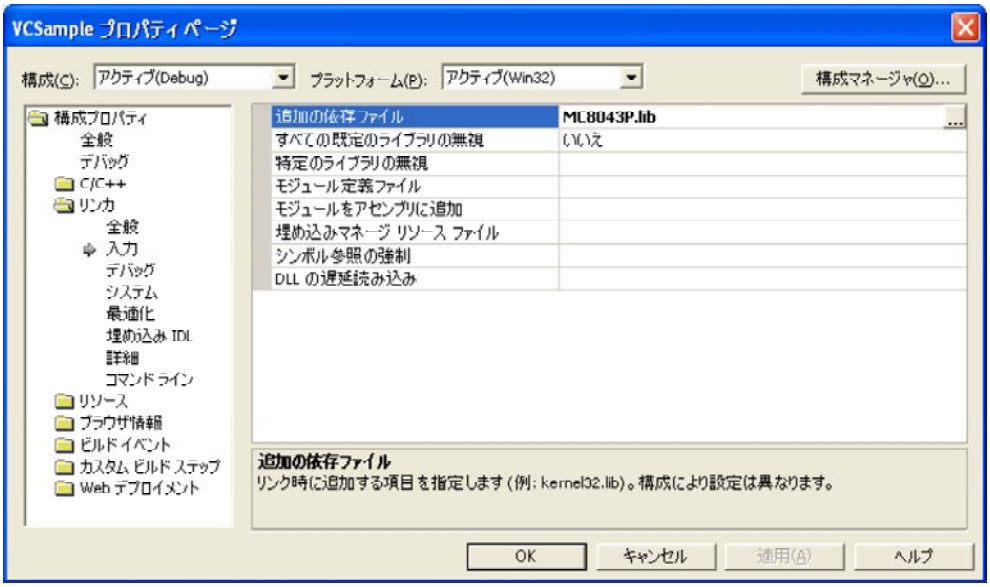


図9.3-2 VC++.NET2003 プロジェクトのプロパティ

■既存のMC8041Pアプリケーションを MC8043Pアプリケーションに変更する場合

注意：MC8043Pの対応ＯＳは、Windows98 Windows2000 WindowsXPです。Windows95,WindowsNTはサポートしていません。

MC8041Pアプリケーションで使用しているMC8041P.libとMC8041P.h（MC8041P用に提供されたファイル）を今回提供のMC8043P.libとMC8043P_DLL.h に差し替え、リビルドする事で、MC8043Pアプリケーションに変更する事ができます。
変更方法は次の通りです。

① ¥Lib¥VC6フォルダに入っている２つのファイルMC8043P.libとMC8043P_DLL.hを開発するアプリケーションのフォルダにコピーしてください。

② VC++の総合開発環境にてMC8043P_DLL.h をご使用のプロジェクトに追加登録し、既存のMC8041P.h をプロジェクトから削除して下さい。また、ソースファイル内でMC8041P.hを#includeしている箇所は、MC8043P_DLL.hの#includeに変更して下さい。
但し、弊社から提供したMC8041P.h に追加修正を行っている場合は、追加した部分のみ残す必要があります。

③ VC++6.0 の場合は、[プロジェクト]－[設定]で「リンク」タブを選択し「オブジェクト／ライブラリモジュール」から MC8041P.libを削除し、MC8043P.libを追加して下さい。（「図9.3-1 VC++6.0 プロジェクトの設定」を参照。）

VC++.NET2003 の場合は、[プロジェクト]－[プロパティ]画面で[リンカ]－[入力]を選択し、「追加の依存ファイル」から MC8041P.libを削除し、MC8043P.libを追加して下さい。（「図9.3-2 VC++.NET2003 プロジェクトのプロパティ」を参照。）

④ MC8041P用に提供されたMC8041P.libとMC8041P.h は使用しませんので、別フォルダに移動しても構いません。

⑤ 9.1.4.1と9.1.4.2 VC++のAPI（MC8041Pドライバで使用していた関数をサポート）の関数をそのまま使用する事ができます。
ボード番号については上記APIの章を参照して下さい。

但し、割込み処理については変更が必要な場合があります。本ドライバでは、割込み発生時にRR3レジスタの値を読み出す場合、ReadRR3関数、あるいはReadRR3_N関数を使用します。InW、InW_N関数で直接RR3レジスタの値を読む事はできません。（理由はReadRR3関数の「注意」参照）
この為、アプリケーションにて、割込み発生時にInW、InW_N関数で直接RR3レジスタの値を読み出している所は、ReadRR3関数、あるいはReadRR3_N関数で値を読み出すように変更して下さい。

⑥ アプリケーションをリビルドして下さい。
リビルドが正常終了した場合、MC8043Pドライバが正常にインストールされたマシンでアプリケーションを実行できます。

9.3.2 ＶＢ６．０でアプリケーションを開発する場合

■新規にMC8043Pアプリケーションを作成する場合

①¥Lib¥VB6 フォルダに入っているMC8043P_DLL.BASファイルを開発するアプリケーションのプロジェクトに標準モジュールとして追加してください。

②9.1.3 ＡＰＩ（ MC8043Pドライバ関数 ）の関数を使用してプログラミングを行って下さい。

まれにデバッグ中など mc8043p.dll にリンクできないときは mc8043p.dll をカレントフォルダにコピーして使用してください。
注意：VB の中で、本ボードに関する割り込みを使用することはできません。

■既存のMC8041PアプリケーションをMC8043Pアプリケーションに変更する場合

注意：MC8043Pの対応ＯＳは、Windows98 Windows2000 WindowsXPです。Windows95,WindowsNTはサポートしていません。

①¥Lib¥VB6 フォルダに入っているMC8043P_DLL.BASファイルを開発するアプリケーションのプロジェクトに標準モジュールとして追加してください。

②既存のMC8041Pアプリケーションソース内に宣言しているMC8041P.DLL関数のDeclare宣言を全て削除して下さい。

◆削除するMC8041P.DLL関数Declare宣言の例

```
Declare Function OpenCard Lib "mc8041p.dll" (ByVal isr As Long) As Long
Declare Function CloseCard Lib "mc8041p.dll" () As Long
```

③9.1.4.3と9.1.4.4 VB6.0のAPI（MC8041Pドライバで使用していた関数をサポート）の関数をそのまま使用する事ができます。
ボード番号については上記APIの章を参照して下さい。
但し、ＶＢでは割り込みをサポートしていませんので、既に割り込み処理が入っている場合は、修正が必要です。

④exeファイルを作成して下さい。
コンパイルが正常終了した場合、MC8043Pドライバが正常にインストールされたマシンでアプリケーションを実行できます。

まれにデバッグ中など mc8043p.dll にリンクできないときは mc8043p.dll をカレントフォルダにコピーして使用してください。

注意：VB の中で、本ボードに関する割り込みを使用することはできません。

9.3.3 ＶＢ.ＮＥＴ２００３でアプリケーションを開発する場合

■新規にMC8043Pアプリケーションを作成する場合

①¥Lib¥VB.NET2003 フォルダに入っているMC8043P_DLL.vbファイルを開発するアプリケーションのプロジェクトに追加してください。

②9.1.3 ＡＰＩ（ MC8043Pドライバ関数 ）の関数を使用してプログラミングを行って下さい。

まれにデバッグ中など mc8043p.dll にリンクできないときは mc8043p.dll をカレントフォルダにコピーして使用してください。
注意：VB の中で、本ボードに関する割り込みを使用することはできません。

■既存のMC8041PアプリケーションをMC8043Pアプリケーションに変更する場合

注意：MC8043Pの対応ＯＳは、Windows98 Windows2000 WindowsXPです。Windows95,WindowsNTはサポートしていません。

①¥Lib¥VB.NET2003 フォルダに入っているMC8043P_DLL.vbファイルを開発するアプリケーションのプロジェクトに追加してください。

②既存のMC8041Pアプリケーションソース内に宣言しているMC8041P.DLL関数のDeclare宣言を全て削除して下さい。

◆削除するMC8041P.DLL関数Declare宣言の例

```
Declare Function OpenCard Lib "mc8041p.dll" (ByVal isr As Integer) As Integer
Declare Function CloseCard Lib "mc8041p.dll" () As Integer
```

③9.1.4.5と9.1.4.6 VB.NET2003のAPI（MC8041Pドライバで使用していた関数をサポート）の関数をそのまま使用する事ができます。
念のため、API 関数の引数と戻り値のデータ型を確認して下さい。ボード番号については上記APIの章を参照して下さい。
但し、ＶＢでは割り込みをサポートしていませんので、既に割り込み処理が入っている場合は、修正が必要です。

④exeファイルを作成して下さい。
コンパイルが正常終了した場合、MC8043Pドライバが正常にインストールされたマシンでアプリケーションを実行できます。

まれにデバッグ中など mc8043p.dll にリンクできないときは mc8043p.dll をカレントフォルダにコピーして使用してください。
注意：VB の中で、本ボードに関する割り込みを使用することはできません。

9.3.4 C#でアプリケーションを開発する場合

MC8043Pアプリケーションでは．ＮＥＴに対応したC#クラスライブラリMc8043pWrap.dllを使用します。

■新規にMC8043Pアプリケーションを作成する場合

① ¥LIB¥Csharpフォルダに入っているファイルMc8043pWrap.dllを、開発するアプリケーションのフォルダにコピーしてください。
② .NETの総合開発環境の[プロジェクト]－[参照の追加]で、コピーしたMc8043pWrap.dllへの参照を追加してください。
③ .NETの総合開発環境の[プロジェクト]－[プロパティ]画面で、構成プロパティのビルドのコード生成のセーフモード以外のコードブロックの許可をtrueに設定してください。
④ ネームスペース　Mc8043pWrap　を追加してください。
⑤ 9.1.3 ＡＰＩ（ MC8043Pドライバ関数 ）の関数を使用してプログラミングを行って下さい。

9.4　プログラミング上の注意点

（１）入力信号フィルタの初期設定

MC8043Pのリミット信号などの各入力信号は、MCX314As内蔵の積分フィルタを使用します。ノヴァ電子より供給されるデバイスドライバでは、パソコン起動時の初期設定において、MCX314Asに拡張モード設定コマンド（60h）を発行して各入力信号に対して、以下のようにフィルタを設定しています。

フィルタ遅延時間：512μsec

各信号のフィルタの有効/無効：

信号名	有効 / 無効
EMG, nLMT+, nLMT-, nIN0, nIN1	有効
nIN2	有効
nINPOS, nALARM	有効
nEXOP+, nEXOP-	有効
nIN3	有効

アプリケーション上で、これらの入力信号フィルタの有効/無効を変更する場合には、MCX314Asのマニュアル6.16節を参照してください。拡張モード設定コマンド（60h）によって変更することが出来ます。次の記述例では、ボード番号０のX,Y,Z,U全軸に対して、上記の初期設定と同じ内容を設定しています。Nmc_ExpModeは拡張モード設定コマンド（60h）を実行しています。

(例１)
```
Nmc_ExpMode(0, AXIS_ALL, 0x5F00, 0x0000);
```

(例２)
```
OutpMC8043P(0, MCX314_WR6, 0x5F00);
OutpMC8043P(0, MCX314_WR7, 0x0000);
OutpMC8043P(0, MCX314_WR0, 0x0F60);
```

注意：
①拡張モード設定コマンド(60h)は、入力信号フィルタの設定(WR6)とともに、自動原点出しの設定(WR7)も行ないます。一方の設定を行なう場合においても必ず両方(WR6,7)に適正値を設定してください。

②アプリケーション上からソフトリセット(WR0/D15に１セット)した場合にも、各入力信号のフィルタは上記のパソコン起動時初期設定と同じ設定がデバイスドライバ内で行なわれます。

（２）パソコンのスタンバイ、休止動作

本ドライバは、パソコンのスタンバイ動作、あるいは休止動作を行った後のドライバの動作を保証していません。
パソコンのスタンバイ動作、あるいは休止動作を行った後に、MC8043Pにアクセスしたい場合は、一度パソコンを再起動させてから行って下さい。

（３）割り込みサポートについて

割り込みは、ＶＣ＋＋、Ｃ＃で開発したアプリケーションでのみサポートしています。
ＶＢで開発したアプリケーションでは割り込みをサポートしていません。

本ドライバでサポートしている割り込みの種類は下記の通りです。

・ＲＲ３レジスタで報告される割り込み全て
・連続補間ドライブで次セグメントのデータと補間ドライブ命令が書き込み可能となった時の割り込み
・ビットパターン補間において、スタックカウンタの値が２から１に変わった時の割り込み

（４）割り込みクリアについて

①ＲＲ３レジスタで報告される割り込みについて

MC8043Pでこの割込が発生した直後、ドライバ内でＲＲ３を読み出し、割り込みをクリアしています。
その後、アプリケーションの割り込み用ユーザー関数が呼び出されます。（ユーザー関数を設定した場合のみ）

②連続補間ドライブで次セグメントのデータと補間ドライブ命令が書き込み可能となった時の割り込みについて

MC8043Pでこの割込が発生した直後、ドライバ内で補間割り込みをクリアしています。
その後、アプリケーションの割り込み用ユーザー関数が呼び出されます。（ユーザー関数を設定した場合のみ）

③ビットパターン補間において、スタックカウンタの値が２から１に変わった時の割り込みについて

MC8043Pでこの割込が発生した直後、ドライバ内で補間割り込みをクリアしています。
その後、アプリケーションの割り込み用ユーザー関数が呼び出されます。（ユーザー関数を設定した場合のみ）

（５）アプリケーションから指定するボード番号について

オープン関数を例にとって説明します。

NO	関数名	関数で指定するボード番号	実際に動作するボード
1	OpenMC8043P	０～９　（ロータリースイッチの値）	指定したボード
2	OpenCard_N	１～１０（ロータリースイッチの値＋１）	指定したボード
3	OpenCard	指定なし	ロータリースイッチの値が０のボード

OpenCard_N 関数で、ボード番号１を指定すると、ロータリースイッチの値が０のボードをオープンします。
OpenCard_N 関数で、ボード番号１０を指定すると、ロータリースイッチの値が９のボードをオープンします。

（６）１枚のボードに対して２つのアプリケーションからの同時アクセス

１枚のボードに対して、２つ以上のアプリケーションから同時にアクセス（オープン等）を行わないで下さい。

（７）既存のMC8041Pアプリケーションを MC8043Pアプリケーションに変更する場合

9.3章 「開発手順」を参照して下さい。

（８）RR3割り込みと補間割り込みを両方使用する場合

RR3で報告される割り込みと補間割り込み(注１)を両方有効にする場合は、割り込みユーザー関数(注２)内で割り込要因を確認する際、RR3の割り込み要因を先に読み出し、その後、補間割り込みが発生しているか確認してください。

例）割り込み発生時の割り込みユーザー関数処理
①ReadEventMC8043P関数(注３)を使用してRR3の割り込み要因を読み出し、RR3で割り込みが発生しているか確認する。
②補間割り込みが発生しているか確認する。(RR0のCNEXT、またはRR0のBPSC1,0を確認)

注１：連続補間ドライブで次セグメントのデータと補間ドライブ命令が書き込み可能となった時の割り込み、または
ビットパターン補間において、スタックカウンタの値が２から１に変わった時の割り込み。

注２：SetEventMC8043P関数で指定したユーザー関数
(MC8041P関数の場合は、OpenCard、またはOpenCard_N 関数で指定したユーザー関数)

注３：MC8041P関数の場合は、ReadRR3、またはReadRR3_N 関数

Windows98 Windows2000 WindowsXP Microsoft Visual C++ Microsoft Visual Basicは米国Microsoft Corporationの登録商標です。

１０．評価ツール

この章では、MC8043P評価ツールについて説明します。

MC8043P（MCX314As）評価ツールプログラムは、弊社ホームページよりダウンロードすることができます。
実行プログラムはTool8043.exeです。評価ツールを実行する前に、MC8043Pドライバをインストールして下さい。

注：この章では、MC8043P評価ツールの概要について説明します。詳細については、Toolフォルダの操作説明書.txtを参照して下さい。

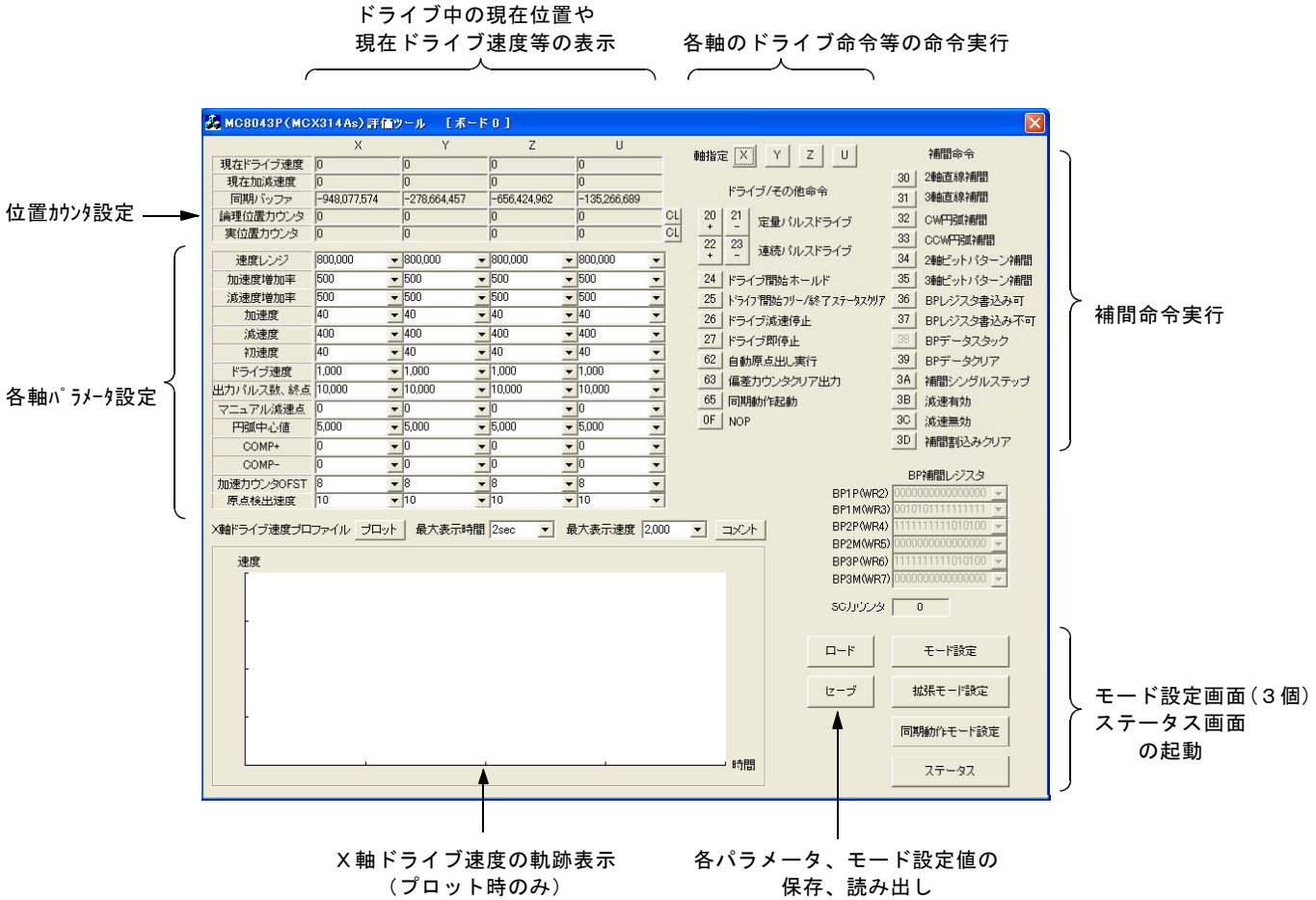
10.1　概要

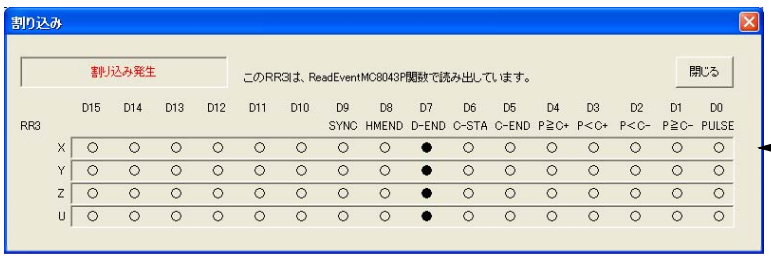
評価ツールを起動すると、ボード番号選択画面が表示されます。そこで、ボード番号（ボードのロータリースイッチ番号(0～9)）を選択し、ＯＫボタンを押すと、メイン画面が表示されます。

メイン画面では、各軸パラメータ設定、ドライブ命令等の命令実行、現在位置・現在速度の表示、割り込み画面表示、パラメータ・モード設定値の保存、読み出し等を行います。また、３種類のモード設定画面やステータス画面を開き、モード設定、ステータス参照等を行う事ができます。

10.2　メイン画面

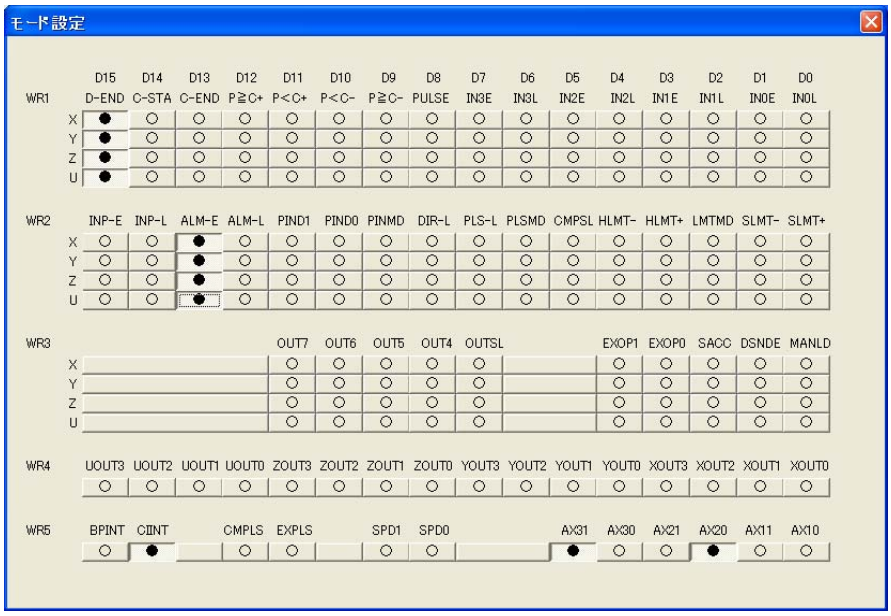
メイン画面では、下図のような操作を行います。





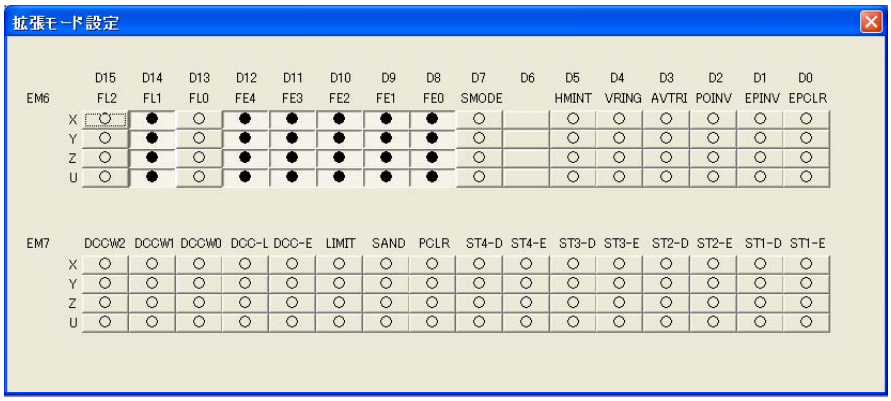
10.3　モード設定画面

モード設定画面では、MCX314AsのWR1～WR5（モードレジスタ、アウトプットレジスタ）を設定します。



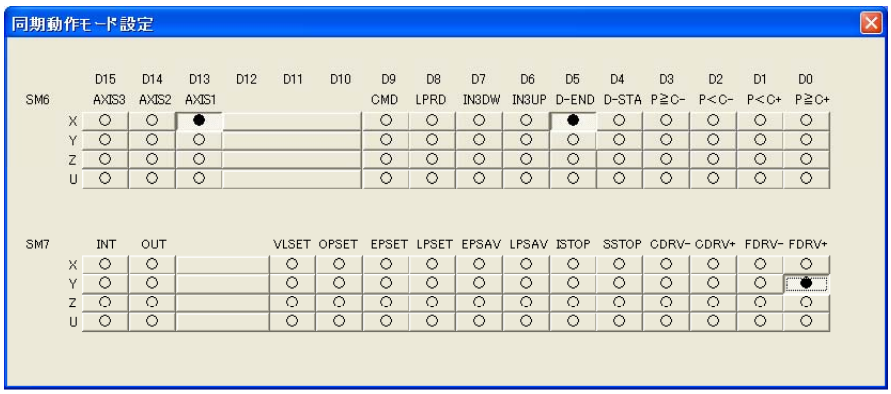
10.4　拡張モード設定画面

拡張モード設定画面では、MCX314Asの拡張モードレジスタ(EM6,EM7)を設定します。



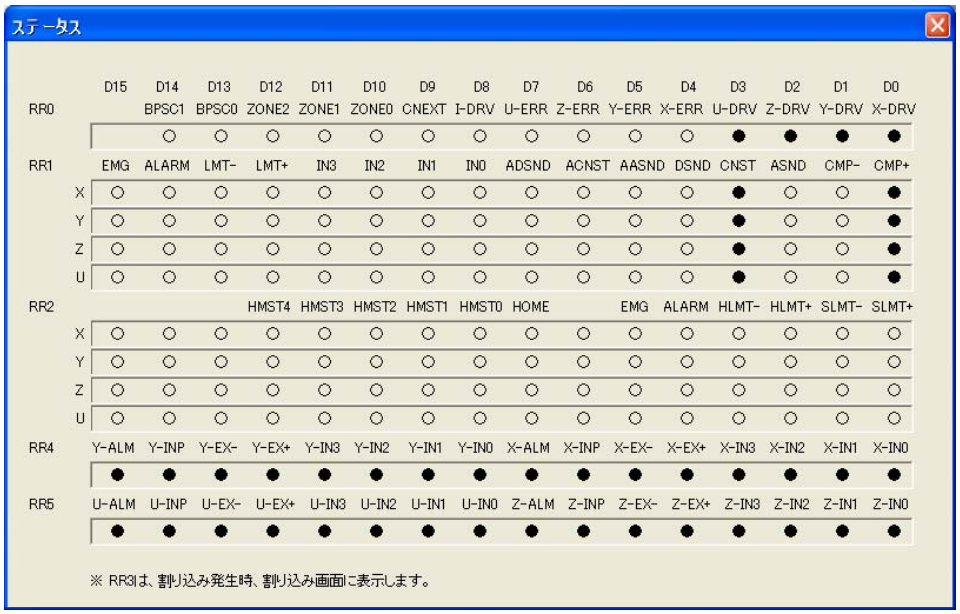
10.5　同期動作モード設定画面

同期動作モード設定画面では、MCX314Asの同期動作モードレジスタ(SM6,SM7)を設定します。



10.6　ステータス画面

ステータス画面では、RR0,RR1,RR2,RR4,RR5（ステータスレジスタ、インプットレジスタ）の現在の値をMCX314Asから読み出し、リアルタイムに画面に表示します。データの読み出しは、50ミリ秒間隔で行います。
RR3については、割り込み発生時、割り込み画面に表示します。



１１．仕様まとめ

● 制御軸　　４軸

ＰＣＩバスインターフェイス

● データビット幅　　１６Bit
● I/O占有アドレス　　１６Byte　アドレスはPnP機能によって任意に決定
● 割り込み　　IRQ　PnP機能によって任意に接続

補間機能

■ ２軸／３軸直線補間

● 補間範囲：　　各軸 -2,147,483,646～+2,147,483,646
● 補間速度：　　1 PPS ～ 4 MPPS
● 補間位置精度：　　±0.5LSB以下（全補間範囲内で）

■ 円弧補間

● 補間範囲：　　各軸 -2,147,483,646～+2,147,483,646
● 補間速度：　　1 PPS ～ 4 MPPS
● 補間位置精度：　　±1 LSB以下（全補間範囲内で）

■ ２軸／３軸ビットパターン補間

● 補間速度：　　1 ～ 4 MPPS（ただしCPUデータセットアップ時間に依存）

■ その他の補間機能　　●任意軸選択可能　●線速一定　●連続補間　●補間ステップ送り（コマンド）

ＸＹＺＵ各軸共通仕様

■ ドライブパルス出力

● 出力回路：　　差動ラインドライバ（ＡＭ２６Ｃ３１）出力
● 出力速度範囲：　　1 PPS　～ 4 MPPS
● 出力速度精度：　　±0.1%以下（設定値に対して）
● 速度倍率：　　1　～ 500
● Ｓ字用加速度/減速度増加率：　　954　～ 62.5×10^{6}PPS/SEC^{2}（倍率=1の時）
　　477×10^{3}　～ 31.25×10^{9}PPS/SEC^{2}（倍率=500の時）
● 加/減速度：　　125　～ 1×10^{6}PPS/SEC　（倍率=1の時）
　　62.5×10^{3}　～ 500×10^{6}PPS/SEC　（倍率=500の時）
● 初速度：　　1　～ 8,000PPS　（倍率=1の時）
　　500PPS　～ 4×10^{6}PPS　（倍率=500の時）
● ドライブ速度：　　1　～ 8,000PPS　（倍率=1の時）
　　500PPS　～ 4×10^{6}PPS　（倍率=500の時）
● 出力パルス数：　　0　～ 4,294,967,295（定量パルスドライブ）

● 速度カーブ：　　定速／対称・非対称直線加減速／対称・非対称放物線Ｓ字加減速ドライブ
● 定量パルスドライブの減速モード：　自動減速（非対称直線加減速も可能）／マニュアル減速
● ドライブ中の出力パルス数、ドライブ速度の変更可能
● 直線加減速定量パルスドライブの三角防止、Ｓ字加減速定量パルスドライブの三角防止機能有り。
● 独立２パルス／１パルス・方向　方式選択可能。
● ドライブパルスの論理レベル選択可能。

■ エンコーダＡ相／Ｂ相／Ｚ相入力

● 入力回路：　高速フォトカプラ入力。差動ラインドライバとの接続可能。
● ２相パルス/アップダウンパルス入力可能。
● ２相パルス　１，２，４逓倍選択可能。

■ 位置カウンタ

● 論理位置カウンタ（出力パルス用）カウント範囲：　-2,147,483,648 ～ +2,147,483,647
● 実位置カウンタ　（入力パルス用）カウント範囲：　-2,147,483,648 ～ +2,147,483,647
常時書き込み、読み出し可能

■ コンペアレジスタ

● COMP+レジスタ位置比較範囲：　-2,147,483,648 ～ +2,147,483,647
● COMP-レジスタ位置比較範囲：　-2,147,483,648 ～ +2,147,483,647
● 位置カウンタとの大小をステータス出力及び信号出力。
● ソフトウェアリミットとして動作可能。

■ **自動原点出し**

● ステップ１（高速原点近傍サーチ)→ステップ２（低速原点サーチ）→ステップ３（低速エンコーダＺ相サーチ）→ステップ４（高速オフセット移動）を順次自動実行。各ステップの有効／無効、検出方向選択可能。

■ **同期動作**

● 起動要因：位置カウンタ≧COMP+変化、位置カウンタ＜COMP+変化、位置カウンタ＜COMP-変化、位置カウンタ≧COMP-変化、ドライブ開始、ドライブ終了、IN3信号↑、IN3信号↓、LP読出し命令、起動命令。

● 動作：　+/-定量パルスドライブ開始、+/-連続パルスドライブ開始、ドライブ減速停止、ドライブ即停止、位置カウンタ値セーブ、位置カウンタセット、出力パルス数セット、ドライブ速度セット、割込み発生。

自軸の要因から任意の他軸動作を起動させるが可能。

■ **割り込み機能**（補間を除く）

● 割り込み発生要因：　１ドライブパルス出力、位置カウンタ≧COMP-変化時、位置カウンタ＜COMP-変化時、位置カウンタ＜COMP+変化時、位置カウンタ≧COMP+変化時、加減速ドライブ中の定速開始時、加減速ドライブ中の定速終了時、ドライブ終了時、自動原点出し終了時、同期動作。

いずれの要因に対しても有効／無効選択可能。

■ **外部信号によるドライブ操作**

● EXPP、EXPM信号により、＋／－方向の定量／連続パルスドライブが可能。

● 入力回路：　フォトカプラ＋IC内蔵積分フィルタ回路。

■ **外部減速停止／即停止信号**

● IN0～3　各軸４点（IN0はエンコーダＺ相入力用）

● 入力回路：　フォトカプラ＋IC内蔵積分フィルタ回路。（IN0は高速フォトカプラ入力）

いずれの信号も有効／無効、論理レベルの選択可能。汎用入力としても使用可能。

■ **サーボモータ用入力信号**

● ALARM（アラーム）、INPOS（位置決め完了）。

● 入力回路：　フォトカプラ＋IC内蔵積分フィルタ回路

いずれの信号も有効／無効、論理レベルの選択可能。

■ **汎用出力信号**

● OUT4～7　各軸４点（汎用出力／ドライブ状態出力　切り換え可能）

● 出力回路：　74LS06出力（オープンコレクタ出力）

■ **ドライブ状態信号出力**

● ASND(加速中)、DSND(減速中)、CMPP(位置≧COMP+)、CMPM(位置＜COMP-)。

ドライブ状態は、ステータスレジスタでも読み出し可能。

■ **オーバランリミット信号入力**

● ＋方向、－方向各１点。

● 入力回路：　フォトカプラ＋IC内蔵積分フィルタ回路

論理レベル選択可能。　アクティブ時、即停止／減速停止選択可能

■ **緊急停止信号入力**

● 全軸でEMGN１点。全軸のドライブパルスを即停止。基板上のジャンパーで論理レベル選択可能。

● 入力回路：　フォトカプラ＋IC内蔵積分フィルタ回路

その他

● **動作温度範囲：**　０℃　～＋４５℃（結露しないこと）

● **電源電圧：**　＋５Ｖ　±５％（消費電流　７００ｍＡmax）

● **外部電源電圧：**　＋１２～２４Ｖ

● **基板外形寸法：**　174.6×106.7mm （コネクタ、金具部は含まず）

● **I/Oコネクタ型式：**　FX2B-100PA-1.27DS （ヒロセ）

● **付属品：**　FX2B-100SA-1.27R （ヒロセ）１.２ｍケーブル付き